UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Rafael Peria de Sene

MODELAGEM DE UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE BASEADO NO OPENUP PARA A FÁBRICA DE SOFTWARE DO LP&D

Alfenas, 06 de junho de 2011.

UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MODELAGEM DE UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE BASEADO NO OPENUP PARA A FÁBRICA DE SOFTWARE DO LP&D

Rafael Peria de Sene

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Alfenas como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Prof. Rodrigo Martins Pagliares

Alfenas, 06 de junho de 2011.

Rafael Peria de Sene

MODELAGEM DE UM PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE* BASEADO NO OPENUP PARA A FÁBRICA DE SOFTWARE DO LP&D

A Banca examinadora abaixo-assinada aprova a monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

Prof. Eduardo Gomes Salgado
Universidade Federal de Alfenas

Prof. Tomas Dias Sant'Ana
Universidade Federal de Alfenas

Prof. Rodrigo Martins Pagliares (Orientador)
Universidade Federal de Alfenas

Alfenas, 06 de junho de 2011.

A meus Pais, meus Irmãos e meus Avós.

AGRADECIMENTO

Muitas foram as pessoas que passaram pela minha vida durante o período de universidade. Cada uma delas deixou uma marca, um gesto, uma palavra ou outra coisa qualquer que me enriqueceu no aprendizado como ser humano e como profissional. Todas essas pessoas merecem ser lembradas e devidamente agradecidas. Infelizmente, não é possível colocar todas neste pequeno trecho de texto, mas isto não minimiza o quão importante foram para que hoje este trabalho pudesse ser escrito.

Agradeço primeiramente meus pais, que não mediram esforços para que eu pudesse concluir meu curso, me apoiando a todo o momento.

Agradeço meus avós, pelas sábias e revigoradoras palavras proferidas nos momentos em que mais precisei.

Agradeço a todos os companheiros de república, em especial ao meu irmão João Paulo, que foram a minha família em Alfenas durante esses cinco anos.

Agradeço ao Marco Antônio, João Batista, Paulo César e Rogério, pessoas que me propiciaram um aprendizado técnico gigantesco.

Agradeço a todos os professores que me guiaram nos primeiros passos na Ciência da Computação, em especial ao meu orientador, Pagliares, por propiciar inigualável aprendizado e por direcionar a construção deste trabalho.

Agradeço a Mari pelo apoio, força, carinho, atenção, dedicação e companheirismo.

Por fim, agradeço ao grandessíssimo amigo Adriano Luis, companheiro fiel dos trabalhos e estudos que nos tomavam dias. Bons dias esses!

"Toda atividade humana pode ser um processo, mas cada um de nós obtém um senso de autovalor daquelas atividades que resultam numa representação ou instância, que pode ser usada ou apreciada por mais de uma pessoa, utilizada repetidamente ou empregada em algum outro contexto não considerado. Isto é, obtemos um sentimento de satisfação pelo reuso de nossos produtos, por nós mesmos ou pelos outros. As pessoas obtêm tanto (ou mais) satisfação de um processo criativo quanto de um produto final. Um artista aprecia tanto as pinceladas quanto o resultado emoldurado. Um escritor aprecia tanto a busca de uma metáfora adequada quanto o livro pronto. Um profissional de software criativo também deveria obter tanta satisfação do processo quanto do produto final."

Roger S. Pressman

RESUMO

Cada vez mais todos os setores da economia tem feito uso (e se tornado dependente) das facilidades providas pelos avanços e descobertas da tecnologia da informação. Tecnologias de previsão do tempo e de localização via satélite, por exemplo, são utilizadas no campo para aumentar a produtividade da lavoura e a eficiência de maquinas colheitadeiras. Softwares que gerenciam toda a cadeia de produção e que controlam maquinas e robôs são cada vez mais comuns nas fábricas. Sistemas de venda e controle de estoque são diferenciais estratégicos indispensáveis ao comércio. Diante deste cenário, produzir e manter software dentro de custos, prazos e critérios de qualidade adequados torna-se requisito obrigatório. Para que isto seja feito de forma consistente, é preciso aliar boas práticas da engenharia de software com um robusto e eficiente processo de desenvolvimento. Processos são úteis porque imprimem consistência e estrutura a um conjunto de atividades, características importantes quando se sabe como fazer algo bem e quer-se garantir que outras pessoas o façam da mesma maneira. Eles podem incluir técnicas obsoletas ou não tirar vantagem das melhores práticas, porém existem processos customizáveis que permitem aliar a boa prática de desenvolvimento, sustentada por um grande arcabouço de métodos, que permitem desenvolver software de qualidade em tempo hábil. O OpenUP é o processo de desenvolvimento de software leve e customizável utilizado neste trabalho como base para a elaboração do processo de desenvolvimento de software da Fábrica de Software do LP&D, esta que apresenta crescimento contínuo do número de projetos de software à ela propostos e, devido a isso, necessita de um processo de desenvolvimento de software que atenda suas necessidades, proporcionando maior controle e organização sobre os projetos que executa.

Palavras-Chave: Engenharia de Software, Processos de Software, OpenUP.

ABSTRACT

Increasingly, all sectors of the economy has made use (and become dependent) of the facilities provided by the advances and breakthroughs in information technology. Technology forecast and satellite location, for example, are used in the field to increase crop yields and efficiency of machine harvesting. Software that manages the entire chain of production and control machines and robots are increasingly common in factories. Sales systems and inventory control are essential to trade strategic differentiators. In this scenario, produce and maintain software within cost, schedule and quality criteria suitable become mandatory requirements. For this to be done consistently, it is necessary to combine best practices of software engineering with a robust and efficient development process. Processes are useful because they print consistency and structure to a set of activities, important features when you know how to do something good and want to make sure others do it alike. They may include techniques obsolete or do not take advantage of best practices, but there are processes that enable customizable ally good development practice, supported by a large framework of methods, which allow developing quality software on time. OpenUP process is lightweight and customizable software development used in this study as a basis for establishing Software Factory's software process development, which shows continuous growth in the number of software projects offered to it. Because of this, it is necessary to implement a software process developing that meets their needs, providing greater control over the organization and the projects it runs.

Keywords: Software Engineering, Software Process, OpenUP.

LISTA DE FIGURAS

FIGURA 1: DIAGRAMA DE CLASSES EM UML 36
FIGURA 2: CICLO DE UM RELEASE EM SCRUM 41
FIGURA 3: CAMADAS DO OPENUP 44
FIGURA 4: CICLO DE VIDA DE UM PROJETO OPENUP 45
FIGURA 5: MARCOS DO OPENUP 46
FIGURA 6: PAPÉIS DO OPENUP 48
FIGURA 7: ATUAL PROCESSO DE DESENVOLVIMENTO DA FÁBRICA DE *SOFTWARE* 57
FIGURA 8: FUNCIONALIDADES DO EPF COMPOSER 59
FIGURA 9: DIAGRAMA DE ATIVIDADES DA FASE DE CONCEPÇÃO 60
FIGURA 10: DIAGRAMA DE ATIVIDADES DA FASE DE ELABORAÇÃO 61
FIGURA 11: DIAGRAMA DE ATIVIDADES DA FASE DE CONSTRUÇÃO 61
FIGURA 12: DIAGRAMA DE ATIVIDADES DA FASE DE TRANSIÇÃO 61
FIGURA 13: WBS DA FASE DE CONCEPÇÃO 62
FIGURA 14: WBS DA FASE DE ELABORAÇÃO 62
FIGURA 15: WBS DA FASE DE CONSTRUÇÃO 63
FIGURA 16: WBS DA FASE DE TRANSIÇÃO 63
FIGURA 17: GERENTE DE PROJETOS E SUAS ATRIBUIÇÕES 64
FIGURA 18: PASSOS DA TAREFA ENCONTRAR, DESCREVER REQUISITOS E DEFINIR VISÃO 64
FIGURA 19: ESTRUTURA CENTRAL DE UM PROCESSO COMO DEFINIDA PELO SPEM 69

LISTA DE TABELAS

TABELA 1: PRINCÍPIOS DOS MÉTODOS ÁGEIS 35
TABELA 2: WORKFLOWS NO RUP 40

LISTA DE ABREVIAÇÕES

AOO	Análise Orientada a Objetos
BPMN	Notação para modelagem de processos de negócio
CMMI	Modelo Integrado de Capacitação e Maturidade
EPF	*Framework* para modelagem de processos
IEEE	Instituto de Engenheiros Elétricos e Eletrônicos
LP&D	Laboratório de Pesquisa e Desenvolvimento
OO	Orientado a Objetos
OpenUP	Processo Unificado Aberto
PMBOK	Corpo de conhecimento em gestão de projetos
POO	Projeto Orientado a Objetos
PU	Processo Unificado
RUP	Processo Unificado Rational
SCRUM	Método de desenvolvimento de software iterativo e incremental
SPEM	Metamodelo para processos de engenharia de software
UMA	Arquitetura de Métodos Unificada
UML	Linguagem de Modelagem Unificada
XP	Método ágil para desenvolvimento de *software*

SUMÁRIO

1 INTRODUÇÃO **25**
1.1 Justificativa e Motivação 26
1.2 Problematização 27
1.2.1 Objetivos Gerais 27
1.2.2 Objetivos Específicos 27
1.3 Organização da Monografia 28
2 FUNDAMENTAÇÃO TEÓRICA **29**
2.1 Considerações Gerais 29
2.2 Estruturas de um Processo 33
2.3 Métodos Ágeis 34
2.4 Processo Unificado 36
2.5 Processo Unificado *Rational* - RUP 38
2.6 SCRUM 41
2.6.1 Definição do *Product Backlog* 42
2.6.2 Planejamento do *Sprint* 42
2.6.3 Andamento do *Sprint* 42
2.6.4 Reuniões Diárias 42
2.6.5 Revisões 43
2.6.6 Monitoramento do Projeto 43
2.7 OpenUP 43
2.7.1 Princípios Fundamentais 46
2.7.2 Papéis 47
2.7.3 Disciplinas e Artefatos 49
2.7.4 Considerações 52
3 MODELAGEM DO PROCESSO **55**
3.1 Análise do Atual Processo 55
3.1.1 Reuniões Semanais 56
3.1.2 Análise Etnográfica 56
3.1.3 Descrição do Atual Processo 57
3.2 Novo Processo de Desenvolvimento 58
3.2.1 Modelagem: Eclipse Process Framework 58
3.2.2 Descrição do Novo Processo 60
3.2.3 Implantação do Processo: primeiros passos 65
4 CONCLUSÕES **67**
5 APÊNDICE **69**
5.1 Apêndice 1: SPEM 69
6 REFERÊNCIAS BIBLIOGRÁFICAS **71**
7 ANEXO **75**
7.1 Anexo 1: Documento Visão "Portal CEAD" 75

1
Introdução

Este capítulo apresenta a introdução ao assunto tratado neste trabalho de conclusão de curso, a justificativa e a motivação para realizá-lo. Apresenta também a problematização, os objetivos gerais e os específicos e a organização desta monografia.

Quando se fornece um serviço ou cria-se um produto, seja desenvolvendo um *software*, escrevendo um relatório ou fazendo uma viagem de negócios, segue-se costumeiramente uma sequencia de etapas para completar um conjunto de tarefas. Estas são realizadas na mesma ordem todas às vezes, por exemplo: não se reboca uma parede sem antes colocar a tubulação necessária; não se assa um bolo sem antes misturar todos os ingredientes. Esse conjunto de tarefas ordenadas pode ser considerado um processo: uma série de etapas que envolvem atividades, restrições e recursos para alcançar a saída desejada (PFLEEGER 2004).

Nos últimos anos, vários processos de desenvolvimento de *software* foram propostos e desenvolvidos. Alguns mais rigorosos com as disciplinas a serem seguidas e os artefatos formais a serem produzidos no ciclo de vida do produto de *software*[1] (PFLEEGER E ATLEE, 2005). Outros destacam os papéis que a flexibilidade poderia ter na produção de *software* de maneira rápida sem desviar o foco da qualidade, de acordo com o mundialmente conhecido "Manifesto Ágil[2]" (AGILE 2010).

Devido à crescente demanda por *software* de qualidade, cada vez mais processos de desenvolvimento de *software* eficazes tornam-se necessários. Em vista disto, é imprescindível que o desenvolvimento e aprimoramento de

[1] Um conjunto de fases do produto que não se sobrepõem, geralmente em ordem sequencial, cujos nomes e quantidades são determinados pelas necessidades de produção e controle da organização (PMBOK 2008).

[2] Disponível em http://www.agilemanifesto.org.

processos de *software* evoluam constantemente de forma a atender as necessidades atuais, bem como sirvam como um arcabouço de conhecimentos para a elaboração de novos processos de desenvolvimento de *software*, mais eficientes, robustos e adaptáveis.

1.1 Justificativa e Motivação

Cada vez mais os setores da economia (primário, secundário e terciário) tem feito uso (e se tornado dependente) das facilidades providas pelos avanços e descobertas da tecnologia da informação. Tecnologias de previsão do tempo e de localização via satélite, por exemplo, são utilizadas no campo para aumentar a produtividade da lavoura e a eficiência de maquinas colheitadeiras. Softwares que gerenciam toda a cadeia de produção e que controlam máquinas e robôs são cada vez mais comuns nas fábricas. Sistemas de venda e controle de estoque são diferenciais estratégicos indispensáveis ao comércio. Diante deste cenário, produzir e manter software dentro de custos, prazos e critérios de qualidade adequados torna-se requisito obrigatório (SOMMERVILLE, 2007).

Para que um software seja desenvolvido de forma consistente, é preciso aliar boas práticas da engenharia de software[3] com um robusto e eficiente processo de desenvolvimento. Diferentes tipos de sistemas necessitam de diferentes processos de desenvolvimento. Por exemplo, um software de tempo real de uma aeronave deve ser completamente especificado antes do inicio do desenvolvimento, enquanto que um sistema de comércio eletrônico a especificação e o desenvolvimento do software podem ser conduzidos paralelamente. O uso de um processo de software inadequado pode reduzir a qualidade ou a utilidade do produto de software a ser desenvolvido e/ou aumentar os custos de desenvolvimento. Este fato leva as organizações que produzem software a usar processos de desenvolvimento que sejam eficientes e que atendam plenamente suas necessidades (SOMMERVILE, 2007).

[3] Engenharia de *Software* é uma disciplina da engenharia relacionada com todos os aspectos da produção de *software*, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que este entrar em produção (SOMMERVILLE, 2007).

Tendo em vista o crescimento contínuo do número de projetos de desenvolvimento de *software* que estão sendo propostos à Fábrica de *Software* do LP&D, faz se necessária à implantação de um processo de desenvolvimento de *software* que atenda suas necessidades, proporcionando maior controle e organização sobre os projetos por ela executados.

1.2 Problematização

Em relação à adaptação de processos de desenvolvimento de software a diferentes organizações, em especial a customização do Processo Unificado Aberto (OpenUP), surge o seguinte questionamento: é possível derivar um processo de desenvolvimento de software a partir do OpenUP, que seja pouco prescritivo e atenda as necessidades da Fábrica de *Software* do Laboratório de Pesquisa e Desenvolvimento da Universidade Federal de Alfenas?

1.2.1 Objetivos Gerais

Definir, modelar e publicar um processo de desenvolvimento de *software* baseado no OpenUP para ser utilizado na Fábrica de Software do Laboratório de Pesquisa e Desenvolvimento (LP&D) da Universidade Federal de Alfenas.

1.2.2 Objetivos Específicos

- Estudar o Processo Unificado Aberto e outros processos de desenvolvimento de software;
- Estudar a ferramenta de modelagem de processos *Eclipse Process Framework*;
- Investigar a maneira como a atual equipe de desenvolvimento da Fábrica de Software executa seu trabalho e qual o fluxo de atividades que eles seguem;
- Capturar os requisitos necessários para modelar o processo de desenvolvimento de software para a Fábrica de Software;
- Modelar e publicar o processo utilizando o EPF *Composer*.

1.3 Organização da Monografia

Esta monografia está organizada da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica na qual este trabalho é embasado, apresentando definições de processo e as descrições do Processo Unificado, do RUP do SCRUM e do OpenUP. O Capítulo 3 contém a metodologia utilizada, a descrição do atual processo da Fábrica de *Software*, da ferramenta de modelagem EPF *Composer* e do novo processo de desenvolvimento de *software*. O Capítulo 4 apresenta as conclusões do que fora feito e exibe algumas considerações que apontam para trabalhos futuros.

2
Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica que embasa esse trabalho. Aqui é feito uma compilação dos principais autores da Engenharia de Software que tem contribuído para o desenvolvimento e melhoria de processos de software.

2.1 Considerações Gerais

Howard Baetjer Jr, em seu livro "*Software as Capital*", faz o seguinte comentário sobre processo de *software* (BAETJER 1998, pág. 85) :

"... Desde que o software, como todo capital, é conhecimento incorporado, e como esse conhecimento está inicialmente disperso, tácito, latente e incompleto na sua totalidade, o desenvolvimento de software é um processo de aprendizado social. O processo é um dialogo no qual o conhecimento, que deve se transformar em software é reunido e incorporado ao software. O processo fornece interação entre usuários e projetistas, entre usuários e ferramentas em desenvolvimento e entre projetistas e ferramentas em desenvolvimento (tecnologia). É um processo iterativo no qual a própria ferramenta serve como meio de comunicação, com cada nova rodada de dialogo explicitando mais conhecimento útil do pessoal envolvido...".

Efetivamente, a elaboração de *software* de computador é um processo de aprendizado, e o resultado, é a incorporação de conhecimentos coletados, destilados e organizados à medida que o processo é conduzido. Processo é o alicerce da engenharia de *software*. É ele que permite o desenvolvimento racional e oportuno de *software*s de computador (PRESSMAN, 2006). Ele pode ser definido para atividades como desenvolvimento, manutenção, aquisição e contratação de *software* (PAULA FILHO, 2009). Processos de *software* formam a base para o controle gerencial de projetos de *software* e estabelece o conteúdo no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos,

documentos, dados, relatórios, formulários, etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas (PRESSMAN, 2006).

O Guia PMBOK® define processo como sendo um conjunto de atividades inter-relacionadas realizadas para obter um conjunto específico de produtos, resultados ou serviços[4] (PMBOK, 2008). Segundo o IEEE[5], um processo é uma sequencia de passos executada com um determinado objetivo (IEEE, 2003).

Para o CMMI[6], um processo é definido quando tem uma descrição que é mantida, ou seja, tem documentação que detalha o que é feito (produto), quando (etapas), por quem (papéis), os itens utilizados (insumos) e os itens produzidos (resultados)(CMMI, 2006). Os processos podem ser definidos com mais ou menos detalhes e suas etapas podem ter ordenação parcial, o que pode permitir paralelismo entre algumas delas (PAULA FILHO, 2009).

Focando no desenvolvimento de *software*, Ian Sommerville define um processo de *software* como um conjunto de atividades que leva à produção de um produto de *software* (SOMMERVILLE, 2007). Roger S. Pressman define processo de *software* como um arcabouço para as tarefas que são necessárias para construir *software* de alta qualidade (PRESSMAN, 2006). Wilson de Paula Filho faz uma analogia interessante, para ele processo é uma receita a ser seguida (PAULA FILHO, 2009).

Processos de *software*s são complexos e como todos os processos intelectuais e criativos dependem de julgamento humano. A existência de um processo de *software* não garante que o *software* será entregue no prazo, de que ele irá satisfazer as necessidades do cliente, ou exibirá os atributos arquiteturais que manterão as características de qualidade em longo prazo. Um processo deve ser acoplado a uma sólida prática de engenharia de *software* e deve ser avaliado para garantir que satisfaça a um conjunto de critérios básicos de

[4] Esta definição é a mesma utilizada pelo CMMI para definir processo (CMMI, 2006).

[5] IEEE é a maior associação profissional do mundo dedicada ao avanço da inovação tecnológica e excelência para o benefício da humanidade (IEEE, 2010).

[6] O CMMI descreve as características de um processo de *software* e os critérios para um processo bem sucedido (CMMI, 2006).

processo que demonstram ser essenciais para uma engenharia de *software* bem sucedida (PRESSMAN, 2006).

Não existe um processo ideal. As organizações devem criar, verificar, validar e aperfeiçoar seus próprios métodos (CMMI, 2006). Várias destas desenvolvem abordagens inteiramente diferentes, adequadas à sua realidade, para o desenvolvimento de *software*. No caso de alguns sistemas, como os sistemas críticos[7], é necessário um processo de desenvolvimento muito bem estruturado. Nos sistemas de negócios, com requisitos que mudam rapidamente, um processo flexível e ágil é provavelmente mais eficaz (SOMMERVILE, 2007).

Existem vários processos de desenvolvimento de *software*, porém algumas atividades fundamentais são comuns a todos eles (SOMMERVILE, 2007):

- ***Especificação:*** define a funcionalidade do *software* e as restrições sobre sua operação.

- ***Projeto e implementação:*** o *software* que atenda a especificação deve ser produzido.

- ***Validação de software:*** o *software* deve ser validado para garantir que ela faça o que o cliente deseja.

- ***Evolução:*** o *software* deve evoluir para atender aos novos requisitos que naturalmente surgirão.

Processos de *software* têm como base modelos de processo genéricos. Esses modelos genéricos não são descrições definitivas de processos de *software*. Ao contrário, são abstrações do processo que podem ser usadas para explicar diferentes abordagens para o desenvolvimento de *software*. Eles podem ser

[7] Sistemas técnicos ou sociotécnicos dos quais as pessoas ou os negócios dependem. Se esses sistemas falharem ao desempenhar seus serviços conforme esperado, podem causar sérios problemas e prejuízos significativos (SOMMERVILLE, 2007).

considerados como *frameworks*[8] de processo que podem ser ampliados e adaptados para criar processos mais específicos de engenharia de *software*. Os modelos genéricos de processos de *software* amplamente utilizados são o modelo em cascata, o modelo de desenvolvimento evolucionário e o modelo de desenvolvimento baseado em componentes. Estes, não são mutuamente exclusivos e comumente são utilizados em conjunto, especialmente para desenvolvimento de sistemas de grande porte (SOMMERVILLE, 2007).

O modelo em cascata considera as atividades fundamentais do processo compreendendo especificação, desenvolvimento, validação e evolução, e as representa como fases de processos separadas, tais como especificação de requisitos, projeto de *software*, implementação, testes, integração e manutenção. Neste modelo não se dá inicio a fase seguinte antes que a anterior tenha sido terminada. As vantagens do modelo cascata consistem na documentação produzida em cada fase e sua aderência a outros modelos de processos de engenharia. Seu maior problema é a divisão inflexível do projeto em estágios distintos. Os compromissos devem ser assumidos no estágio inicial do processo, o que torna difícil reagir às mudanças de requisitos. Este modelo pode ser usado quando os requisitos forem bem compreendidos e houver pouca probabilidade de mudanças radicais durante o desenvolvimento do sistema (PRESSMAN, 2006).

O modelo de desenvolvimento evolucionário intercala as atividades de especificação, desenvolvimento e validação. Um sistema inicial é desenvolvido rapidamente baseado em especificações abstratas. Este sistema é, então, refinado com as entradas do usuário para produzir um sistema que satisfaça suas necessidades. Esta abordagem é mais eficaz do que a abordagem em cascata na produção de sistemas que atendam às necessidades imediatas dos usuários. A vantagem de um processo de *software* baseado na abordagem evolucionária é que a especificação pode ser desenvolvida de forma incremental. À medida que os usuários compreendem melhor seu problema, esse conhecimento é repassado para o desenvolvimento do *software* (SOMMERVILLE, 2007).

[8] *Framework* pode ser definido sob dois aspectos: estrutura e função. Quanto à estrutura, um *framework* é a reutilização do *design* de parte ou totalidade de um sistema representado por classes abstratas e pela forma como instâncias destas classes interagem (JOHNSON, 1997).

A engenharia de *software* baseada em componentes baseia-se na existência de um numero significativo de componentes reusáveis. O processo de desenvolvimento do sistema foca a integração desses componentes, em vez de desenvolvê-los a partir do zero. Ela tem a vantagem óbvia de reduzir a quantidade de *software* a ser desenvolvido e, desta maneira, reduzir os custos e riscos. Isto também leva a uma entrega mais rápida de *software*. No entanto, compromissos com os requisitos são inevitáveis e isso pode levar a um sistema que não atenda às reais necessidades dos usuários. Além disso, algum controle sobre a evolução do sistema será perdido se novas versões dos componentes reusáveis não estiverem sob controle da organização que os utiliza (SOMMERVILLE, 2007).

Embora não exista um processo de *software* ideal, existe espaço para o aprimoramento (SOMMERVILE, 2007). Os processos podem incluir técnicas obsoletas ou não tirar vantagem das melhores práticas, porém existem processos customizáveis que permitem aliar a boa prática de desenvolvimento, sustentada por um grande arcabouço de métodos que permitem desenvolver *software* de qualidade em tempo hábil. Este, fator de grande importância para o desenvolvimento de uma aplicação levando-se em consideração a demanda por sistemas baseados em *software* para o mundo dos negócios que operam atualmente em ambiente global, sujeito a rápidas mudanças e tendo que responder a novas oportunidades e mercados, mudanças de condições econômicas e ao surgimento constante de novos produtos e serviços. Neste cenário, é praticamente impossível derivar um conjunto completo de requisitos de *software* estáveis. Os processos de desenvolvimento de *software* baseados em especificações completas de requisitos, projeto, construção e teste de sistema não estão voltados para o desenvolvimento rápido de *software*.

2.2 Estruturas de um Processo

Todo processo (uma organização lógica de pessoas, materiais, energia, equipamentos e procedimentos dentro de atividades de trabalho designadas a produzir um resultado específico (PALL, 1987), possui a seguinte relação entre os elementos que o compõem: possui uma *Work Breakdown Structure*

(decomposição orientada à entrega do trabalho a ser executado para atingir os objetivos do projeto e criar as entregas necessárias). Esta é dividida em *Fases* (um conjunto de atividades do projeto, relacionadas de forma lógica, que geralmente culminam na entrega de algum item importante). Cada fase é realizada utilizando-se um determinado número de *Iterações* (um período de tempo definido em que se produz um incremento de um produto), e é composta por *Atividades* (um conjunto de tarefas relacionadas para cumprir um determinado objetivo). Cada atividade possui um conjunto de *Tarefas (*descrevem como executar o trabalho). Toda tarefa pertence a uma *Disciplina (*agrupamentos de tarefas que compartilham de um mesmo propósito), é executada por um *Ator*, produz/consome *Produtos de trabalho*, é composta por *Passos* e usa *Orientações*. Produtos de trabalho usam exemplos/orientações e são baseados em *Modelos*.

2.3 Métodos Ágeis

Processos de desenvolvimento ágil de *software* são projetados para criar *software* útil rapidamente. O *software* não é desenvolvido e disponibilizado integralmente, mas em uma série de incrementos, e cada incremento inclui uma nova funcionalidade do sistema (PRESSMAN, 2006).

Provavelmente o método ágil mais conhecido é o *Extreme Programming* (XP) (BECK, 2000). Nele, todos os requisitos são expressos como cenários (chamados histórias do usuário), que são implementados diretamente como uma série de tarefas. O cliente está intimamente envolvido na especificação e priorização dos requisitos de sistema, ele é parte da equipe de desenvolvimento e discute cenários com os outros membros. Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes da escrita do código. Todos os testes devem ser executados com sucesso quando um novo código é integrado ao sistema (SOMMERVILLE, 2007).

Outro método de desenvolvimento ágil bastante utilizado é o SCRUM (COHN, 2011)(Seção 2.4), que é um processo ágil e leve que pode ser utilizado para gerenciar e controlar o desenvolvimento de *software* utilizando práticas iterativas e incrementais. Baseado no XP e no Processo Unificado (PU) (Seção

2.2) produz os benefícios do desenvolvimento ágil com a vantagem de ser uma implementação bem simples. Ele aumenta significativamente a produtividade e reduz o tempo para obter resultados, pois facilita a adaptação a processos empíricos[9] de desenvolvimento de sistemas.

Embora os métodos ágeis sejam todos baseados na noção de desenvolvimento e entregas incrementais, eles propõem processos diferentes para conseguir isso. Contudo, compartilham um conjunto de princípios e, portanto, têm muito em comum (SOMMERVILLE, 2007). Esses princípios são mostrados na Tabela 1.

Princípio	Descrição
Envolvimento do cliente	Clientes devem ser profundamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar as iterações do sistema.
Entrega Incremental	O *software* é desenvolvido em incrementos e o cliente especifica os requisitos a serem incluídos em cada incremento.
Pessoas, não processo	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Os membros da equipe devem desenvolver suas próprias maneiras de trabalhar sem processos prescritivos.
Aceite as mudanças	Tenha em mente que os requisitos do sistema vão mudar, por isso projete o sistema para acomodar essas mudanças.
Mantenha a simplicidade	Concentre-se na simplicidade do *software* que está sendo desenvolvido e do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.

Tabela 1: Princípios dos métodos ágeis (Sommerville 2007, pág. 263)

[9] Baseia-se exclusivamente na experiência e na observação.

2.4 Processo Unificado

Durante a década de 1980 e inicio da de 1990, métodos e linguagens de programação orientada a objetos (OO) ganharam notoriedade por toda a comunidade de engenharia de *software*. Uma grande variedade de métodos de análise orientados a objetos (AOO) e projeto orientado a objetos (POO) foram propostos durante o mesmo período de tempo. No inicio da década de 1990, James Rumbaugh (Rumbaugh, 1991), Grady Booch (Booch, 1994) e Ivar Jacobson (Jacobson, 1992) começaram a trabalhar em um "método unificado" que combinaria as melhores características de cada um dos seus métodos individuais e adotaria características adicionais propostas por outros especialistas. Deste trabalho originou-se a UML (*Unifield Modeling Language*), uma linguagem unificada de modelagem que contém uma notação robusta para a modelagem e desenvolvimento de sistemas OO. A UML fornece a tecnologia necessária para apoiar a prática de engenharia de *software* orientada a objetos, mas não fornece o arcabouço de processo para guiar as equipes de projeto na aplicação da tecnologia (PRESSMAN, 2006). A Figura 1 mostra um exemplo de modelagem feita utilizando UML.

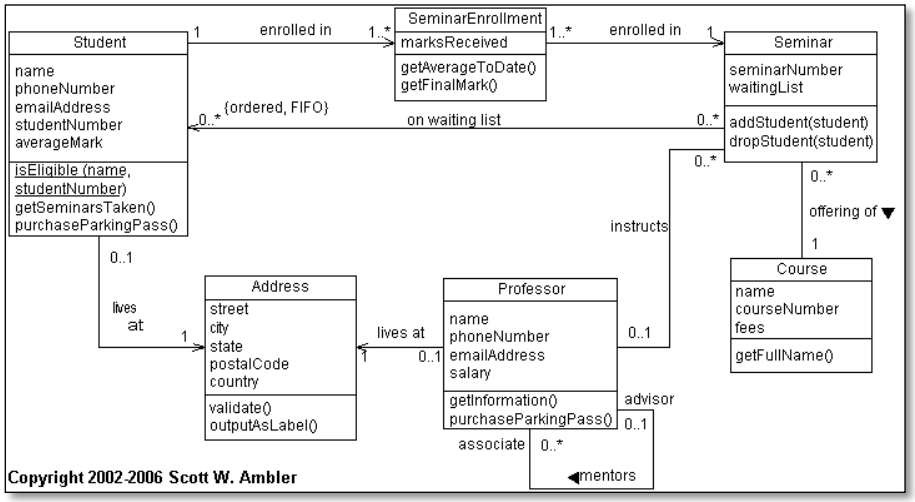

Figura 1: Diagrama de Classes em UML (AGILEDATA, 2011)

Ao longo dos cinco anos seguintes a concepção da UML, Jacobson, Rumbaugh e Booch trabalharam no desenvolvimento do Processo Unificado (PU). Este surgiu como um processo iterativo para o desenvolvimento de

software orientado a objetos promovendo boas práticas amplamente reconhecidas como melhores. É flexível, aberto, incentiva a inclusão de práticas interessantes de outros métodos iterativos e combina ciclo de vida iterativo e desenvolvimento guiado por risco em uma descrição de processo coesa e bem documentada (LARMAN, 2007). Os três aspectos que caracterizam o processo unificado são capturados em três conceitos chave abaixo descritos:

- **Direcionado a casos de uso:** um caso de uso conta uma história sobre como um usuário final (alguém desempenhando um entre vários papéis possíveis) interage com o sistema sob um conjunto específico de circunstâncias. A história pode ser um texto narrativo, um delineamento das tarefas ou interações, ou uma representação diagramática (PRESSMAN, 2010).

- **Centrado na arquitetura:** arquitetura de *software* é o *framework* fundamental para a estruturação do sistema. Ela é influenciada por muitos fatores, tais como a plataforma de *software* sobre a qual o sistema vai ser executado (sistema operacional, sistema gerenciador de banco de dados, protocolos para comunicação em rede), porções de código reusáveis (por exemplo, um *framework* para construção de interface gráfica com o usuário), sistemas legados e requisitos não funcionais (desempenho, proteção, disponibilidade, etc.) (SOMMERVILLE, 2007).

- **Iterativo e incremental:** o desenvolvimento é baseado em uma série de miniprojetos curtos, de duração fixa (por exemplo, três semanas) chamados iterações. O produto de cada um é um sistema parcial, executável, testável e integrável. Cada iteração inclui suas próprias atividades de análise de requisitos, projeto, implementação e testes. O sistema cresce incrementalmente ao longo do tempo, iteração após iteração (LARMAN, 2007).

O ciclo de vida de um projeto[10] PU é composto por quatro fases: Concepção, Elaboração, Construção e Transição.

[10] Um conjunto de fases do projeto, geralmente em ordem sequencial, cujos nomes e quantidades são determinados pelas necessidades de controle da organização ou organizações envolvidas no projeto. Um ciclo de vida pode ser documentado com uma metodologia (PMBOK, 2008).

Concepção é um passo inicial curto para estabelecer uma visão comum e o escopo básico do projeto. Sua finalidade não é definir todos os requisitos ou criar planos e estimativas confiáveis e sim fazer uma investigação suficiente para formar uma opinião racional e justificável da finalidade geral e da viabilidade do novo sistema em potencial, para então decidir se vale a pena investir em uma exploração mais profunda (LARMAN, 2007).

Elaboração é a fase durante a qual se faz uma investigação séria, implementa (programa e testa) a arquitetura central, esclarece a maioria dos requisitos e ataca os problemas de alto risco ("risco" inclui valor de negócio). Ela não é uma fase de projeto ou uma fase na qual os modelos são desenvolvidos completamente como forma de preparação para a implementação na fase de construção (LARMAN, 2007).

A Construção desenvolve ou adquire os componentes de *software* que vão tornar cada caso de uso operacional para os usuários finais. Para isso, os modelos de análise e projeto que foram iniciados durante a fase de elaboração são completados de modo a refletir a visão final do incremento de *software* (PRESSMAN, 2006).

Finalmente na Transição, o *software* é colocado em ambiente de produção para testes beta[11] e *feedback* sobre possíveis defeitos e modificações. Além disso a equipe de *software* cria informações de apoio necessárias que precisam ser entregues (por exemplo, manuais de usuário, guias de solução de problemas e procedimentos de instalação) (PRESSMAN, 2006).

2.5 Processo Unificado *Rational* - RUP

Rational Unified Process (RUP) é um processo de engenharia de *software* que fornece uma abordagem disciplinada para assumir tarefas e responsabilidades dentro de uma organização de desenvolvimento, cujo objetivo é assegurar a

[11] Ação controla de teste na qual o *software* é usado por usuários finais reais com a intenção de descobrir defeitos e deficiências (PRESSMAN, 2006).

produção de *software* de alta qualidade dentro de prazos e orçamentos previsíveis (KRUCHTEN, 2003). Derivado dos trabalhos sobre UML[12] e do Processo Unificado (Seção 2.1), ele traz elementos de todos os modelos genéricos de processo, apoia a iteração e ilustra boas práticas de especificação e projeto (SOMMERVILLIE, 2007). Ele captura seis das melhores práticas no desenvolvimento de *software* de forma satisfatória para uma grande faixa de projetos e organizações (KRUTCHTEN, 2003). As melhores práticas abordadas são as seguintes (SOMMERVILLE, 2007):

- **Desenvolver o *software* iterativamente:** planejar os incrementos de *software* com base nas prioridades do cliente e desenvolver e entregar o mais cedo possível às características de sistema de maior prioridade no processo de desenvolvimento.

- **Gerenciar Requisitos:** documentar explicitamente os requisitos do cliente e manter acompanhamento das mudanças desses requisitos. Analisar o impacto das mudanças no sistema antes de aceitá-las.

- **Usar arquiteturas baseadas em componentes:** Estruturar a arquitetura do sistema com componentes, reduzindo a quantidade de *software* a ser desenvolvido e, consequentemente, reduzir custos e riscos.

- **Modelar *software* visualmente:** usar modelos gráficos de UML para apresentar as visões estática e dinâmica do *software*.

- **Verificar a qualidade do *software*:** garantir que o *software* atenda aos padrões de qualidade da organização.

- **Controlar as mudanças do *software*:** gerenciar as mudanças do *software*, usando um sistema de gerenciamento de mudanças, procedimentos e ferramentas de gerenciamento de configuração.

O RUP é um modelo constituído por quatro fases do processo de *software*, relacionadas mais estritamente aos negócios do que a assuntos técnicos (SOMMERVILLE, 2007). As quatro fases do RUP são as mesmas quatros fases descritas no Processo Unificado (Seção 2.1). As atividades que ocorrem durante

[12] É uma linguagem visual para especificar, construir e documentar os artefatos do sistema (LARMAN, 2007).

o processo de desenvolvimento são chamadas de *workflows*. Existem nove *workflows* principais, exibidos na Tabela 2.

Embora o RUP não seja um processo adequado a todos os tipos de desenvolvimento de *software*, ele representa uma geração de processos genéricos. A mais importante inovação é a separação de fases e workflows, e o reconhecimento de que a implantação de *software* no ambiente do usuário é parte do processo. As fases são dinâmicas e tem objetivos. Os workflows são estáticos e constituem atividades técnicas que não estão associadas a uma única fase, mas podem ser utilizados ao longo do desenvolvimento para atingir os objetivos de cada fase (SOMMERVILLE, 2007).

Workflow	**Descrição**
Modelagem de Negócios	Os processos de negócio são modelados usando casos de uso de negócios.
Requisitos	Os agentes que interagem com o sistema são identificados e os casos de uso são desenvolvidos para modelar os requisitos do sistema.
Análise e Projeto	Um modelo de projeto é criado e documentado usando modelos de arquitetura, modelos de componente, modelos de objetos e modelos de sequencia.
Implementação	Os componentes de sistema são implementados e estruturados em subsistemas de implementação. A geração automática de código com base os modelos de projeto ajuda a acelerar esse processo.
Teste	O teste é um processo iterativo realizado em conjunto com a implementação. O teste de sistema segue o término da implementação.
Implantação	Uma versão do produto é criada, distribuída aos usuários e instalada no local de trabalho.
Gerenciamento de Configuração e Mudança	Este workflow de apoio gerencia mudanças no sistema.
Gerenciamento de Projetos	Este workflow de apoio gerencia o desenvolvimento do sistema.
Ambiente	Este workflow está relacionado à disponibilização de ferramentas apropriadas de *software* para a equipe de desenvolvimento.

Tabela 2: Workflows no RUP (SOMMERVILLE, 2007)

2.6 SCRUM

SCRUM é um método ágil de desenvolvimento de *software*. Baseia-se em ciclos com período de tempo definido chamados *Sprints*, onde trabalha-se para alcançar objetivos bem definidos. Estes objetivos são representados no *Product Backlog*, uma lista de itens a fazer constantemente atualizada e repriorizada (SCRUMALLIANCE, 2010). A Figura 2 ilustra o ciclo de um release no SCRUM.

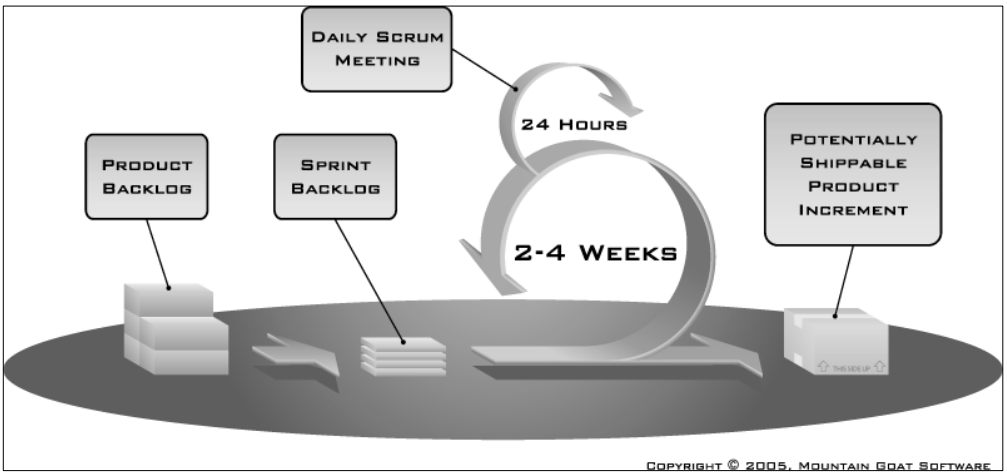

Figura 2: Ciclo de um release em SCRUM (MOUNTAIN GOAT, 2011)

Existem três papéis principais que o compõem: a *Equipe*, *Product Owner* e *Scrum Master*.

- **Equipe**: Responsável por entregar soluções. Geralmente é formada por um grupo pequeno, multifuncional e que trabalha de forma autogerenciada e auto-organizada.

- **Product Owner**: Responsável pela visão de negócios do projeto. Cabe a ele definir e priorizar o *Product Backlog*. Geralmente é o papel desempenhado pelo cliente ou por um gerente de produtos da empresa que desenvolve o *software*.

- **Scrum Master**: é uma mistura de gerente, facilitador e mediador. Seu papel é remover obstáculos da equipe e assegurar que as práticas de *Scrum* estejam sendo executadas com eficiência.

2.6.1 Definição do *Product Backlog*

Todas as funcionalidades ou mudanças no produto são definidas pelo *Product Owner* no *Product Backlog*. Esta lista é priorizada para refletir a necessidade dos clientes ou demandas do mercado. Os itens do topo possuem prioridade maior do que itens no final da lista.

2.6.2 Planejamento do *Sprint*

Ocorre em duas partes, cada uma delas com *time-box*[13] definido. Na primeira parte do planejamento, o *Scrum Master* reúne-se com o *Product Owner* para verificar qual é o *Product Backlog*. Na segunda parte, o *Scrum Master* reúne-se com a *Equipe* para estimar o esforço necessário para os itens do *Product Backlog* (usando estimativa ágil como *Planning Poker,* por exemplo) e para planejar o *Sprint*.

2.6.3 Andamento do *Sprint*

Durante o *Sprint*, os itens do *Product Backlog* que devem ser entregues são tratados em um artefato conhecido como *Sprint Backlog*, que contém os itens do *Product Backlog* subdivididos em tarefas menores. As tarefas são de responsabilidade da *Equipe*, que tem autonomia para decidir como elas devem ser executadas.

2.6.4 Reuniões Diárias

O *Scrum Master* se reúne diariamente com a *Equipe* sempre em um mesmo horário. Nesta reunião, cada membro da equipe responde três perguntas básicas:

- O que foi feito ontem?
- O que se pretende fazer hoje?

[13] Período de tempo.

- Quais são os impedimentos que estão atrapalhando a execução das tarefas?

2.6.5 Revisões

No final do *Sprint* a *Equipe* demonstra os resultados para o *Product Owner* e demais interessados em uma reunião chamada *Sprint Review*, de forma que os itens do *Backlog* desenvolvidos sejam validados e então possa se iniciar um novo *Sprint*. A equipe também faz uma reunião de auto avaliação no final do *Sprint* chamada *Sprint Restrospective.* Nela discute-se o que aconteceu durante a iteração finalizada, identificando o que funcionou bem e o que pode ser melhorado.

2.6.6 Monitoramento do Projeto

Para reportar o andamento do projeto usa-se um *Burndown Chart,* que é um gráfico onde o eixo x é a linha de tempo do projeto e o eixo y é o número de pontos a serem vencidos, que representam os pesos das funcionalidades potencialmente implantáveis do *Product Backlog.*

2.7 OpenUP

O OpenUP é um processo de desenvolvimento de *software* leve, derivado do RUP (Seção 2.2) e inspirado por práticas ágeis advindas do SCRUM (Seção 2.3). Ele aplica uma abordagem iterativa e incremental dentro de um ciclo de vida estruturado e abraça uma filosofia pragmática e ágil que foca na natureza colaborativa do desenvolvimento de *software*. É um processo objetivo e independente de ferramenta, que pode ser estendido para direcionar uma grande variedade de projetos. É considerado um processo mínimo, completo e extensível, valorizando a colaboração entre a equipe e os benefícios aos interessados, ao invés da formalidade e entrega de itens desnecessários (OPENUP, 2010). O processo pode ser facilmente entendido através de três

camadas: Microincrementos, Ciclo de Vida de Iteração e Ciclo de Vida do Projeto, exibidas na Figura 3.

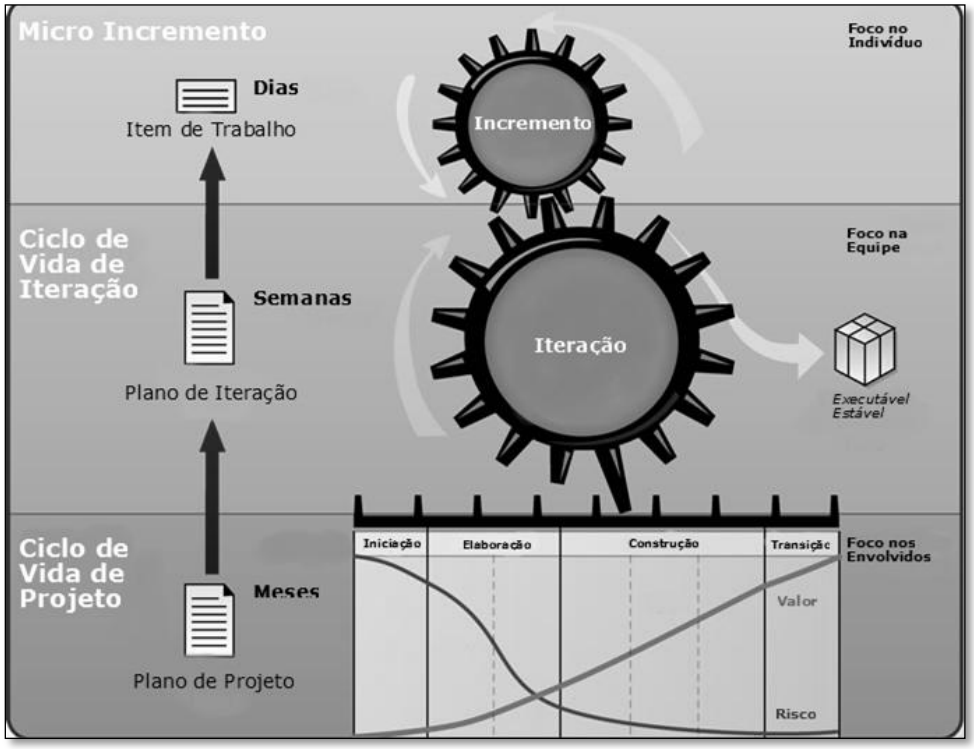

Figura 3: Camadas do OpenUP (OpenUP, 2010)

O esforço pessoal em um projeto OpenUP está organizado em microincrementos. Eles representam pequenas unidades de trabalho que produzem um passo do progresso do projeto, constante e mensurável (normalmente medido em horas ou dias). O processo aplica a colaboração intensiva à medida que o sistema é desenvolvido incrementalmente, por uma equipe comprometida e auto-organizada. Estes microincrementos fornecem um ciclo de *feedback* extremamente curto, que direciona decisões adaptativas durante cada iteração (OPENUP, 2010).

O OpenUP divide o projeto em iterações planejadas e com intervalos de tempo definidos, normalmente medidos em semanas. As iterações direcionam a equipe na entrega incremental de valor aos *stakeholders*[14] de forma previsível. O plano de iteração define o que deve ser entregue durante a iteração e o

[14] Papel que deve ser executado por qualquer pessoa que é, ou será afetada pelo resultado do projeto (OPENUP, 2010).

resultado é uma construção demonstrável ou implantável. As equipes OpenUP se auto-organizam para definir como atingir os objetivos da iteração e entregar o resultado. Elas fazem isso definindo e distribuindo tarefas detalhadas de uma lista de itens de trabalho. O OpenUP usa um ciclo de vida de iteração que estrutura como os microincrementos são aplicados para entregar construções estáveis e coesas do sistema, que progridem incrementalmente na direção dos objetivos da iteração. Ele estrutura o **ciclo de vida do projeto** em quatro fases: Concepção *(Inception),* Elaboração *(Elaboration),* Construção *(Construction)* e Transição *(Transition),* as mesmas quatros fases descritas no Processo Unificado (Seção 2.1) e Rational Unified Process (Seção 2.2). Estas quatro fases são ilustradas na Figura 4.

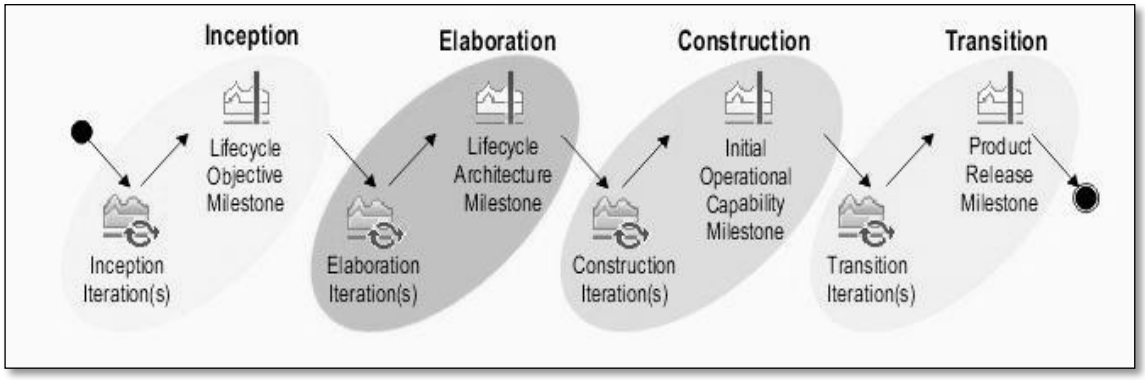

Figura 4: Ciclo de vida de um projeto OpenUP (OpenUP, 2010)

O ciclo de vida de projeto fornece aos *stakeholders* e à equipe de projeto, visibilidade e pontos de decisão durante o projeto. Isto permite uma efetiva supervisão para tomar decisões de "prosseguir ou parar" em momentos apropriados. Um plano de projeto define o ciclo de vida, e o resultado final é uma aplicação passível de ser utilizada. Em cada uma das quatro fases do ciclo de vida de um projeto existe um marco (*Milestone*) que determina formalmente o final de uma fase. Eles proporcionam um ponto de verificação para identificar se o projeto está pronto para prosseguir para a fase seguinte. Cada fase corresponde a um período de tempo entre dois marcos. Quando um marco não é cumprido, mais iterações podem ser realizadas antes da fase ser considerada completa. O final da fase de Concepção (*Inception*) é o primeiro grande marco do ciclo de vida do projeto. Neste ponto, examina-se a relação custo/benefício e decide-se prosseguir ou cancelar o projeto. O fim da fase de Elaboração (*Elaboration*) é o segundo marco importante. Neste ponto, traça-se um mapa dos

requisitos, examinam-se os objetivos do sistema e o escopo, escolhe-se a arquitetura e averiguam-se os principais riscos. O marco é alcançado quando a arquitetura for validada. O final da fase de Construção (*Construction*) é o terceiro marco importante. Neste ponto, o produto está pronto para ser entregue à equipe de transição. O final da fase de Transição (*Transition*) é a quarto marco importante. Neste ponto, averigua-se se os objetivos foram atingidos e se um novo ciclo de desenvolvimento deve ser iniciado (OPENUP, 2010). Os marcos do OpenUp são ilustrados na Figura 5.

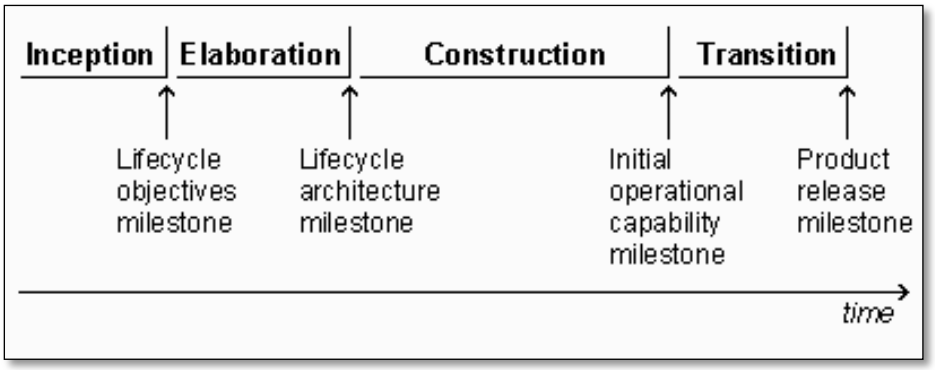

Figura 5: Marcos do OpenUP (OpenUP, 2010)

2.7.1 Princípios Fundamentais

O OpenUP está baseado em quatro princípios fundamentais mutuamente suportados:

- **Equilibrar as prioridades concorrentes para maximizar o benefício aos Stakeholders:** promover práticas que permitam aos participantes do projeto e aos *stakeholders* desenvolver uma solução que maximize os benefícios para o *stakeholder*, e que seja compatível com as restrições impostas ao projeto.

- **Colaborar para alinhar os interesses e compartilhar o entendimento:** promover práticas que estimulem um ambiente de equipe saudável, permitam a colaboração e desenvolvam uma compreensão compartilhada do projeto.

- **Focar na arquitetura, o mais cedo possível, para reduzir o risco e organizar o desenvolvimento:** promover práticas que permitam à equipe focar na arquitetura para reduzir o risco e organizar o desenvolvimento.

- **Evoluir para continuamente obter *feedback* e promover melhorias:** promover práticas que permitam à equipe obter *feedback* dos *stakeholders*, o mais cedo possível e de forma contínua, e demonstrar valor incremental para eles.

2.7.2 Papéis

Software é criado por pessoas com diferentes interesses e habilidades. As pessoas são o maior patrimônio de uma organização. Elas representam o capital intelectual da empresa (SOMMERVILLE, 2007). As pessoas nos papéis executam as tarefas que consomem e produzem os artefatos. Um ambiente de equipe saudável permite uma colaboração efetiva e exige uma cultura engajada na criatividade e na mudança positiva. Os papéis são a face humana do processo de desenvolvimento de *software*. Eles não representam responsabilidades individuais sobre as tarefas ou artefatos e não se limitam a descrever o principal executor de uma tarefa, pelo contrário, incluem uma perspectiva sobre colaboração fornecendo executores adicionais para cada tarefa. Executar um ou mais papéis pode ajudar as equipes a exprimirem diferentes pontos de vista para criar uma solução. Esta perspectiva sobre os papéis fortalece a nova geração de processos de desenvolvimento de *software*, mais focada na interação das pessoas. Ninguém constrói um bom *software* sozinho, mas uma equipe trabalhando junto pode fazer coisas extraordinárias (OPENUP, 2010).

Os papéis do OpenUp, estão descritos abaixo e representados na Figura 6.

- ***Arquiteto:*** responsável por definir a arquitetura de *software*, incluindo a tomada das principais decisões técnicas que orientam todo o *design* e a implementação do projeto.

- ***Gerente de Projeto:*** conduz o planejamento do projeto, coordena as interações com os *stakeholders* e mantêm a equipe de projeto focada em alcançar os objetivos do projeto.

- ***Analista:*** representa os interesses do cliente e do usuário final recolhendo informações dos *stakeholders* para entender o problema a ser resolvido, capturando os requisitos e definindo suas prioridades.

- ***Testador:*** responsável pelas principais atividades do esforço de teste. Estas atividades incluem identificar, definir, implementar e conduzir os testes necessários, bem como registrar e analisar os resultados dos testes.

- ***Qualquer papel:*** Qualquer um em uma equipe pode atuar neste papel executando diversas tarefas.

- ***Desenvolvedor:*** responsável por desenvolver uma parte do sistema, incluindo a construção de seu desenho de forma que ele atenda a arquitetura e possivelmente a prototipagem da interface de usuário, e então implementar, executar o teste de unidade e integrar os componentes que são parte da solução.

- ***Stakeholder:*** representa grupos de interessados cujas necessidades devem ser satisfeitas pelo projeto. É um papel que pode ser executado por qualquer um que seja (ou potencialmente possa ser) afetado pelo resultado do projeto.

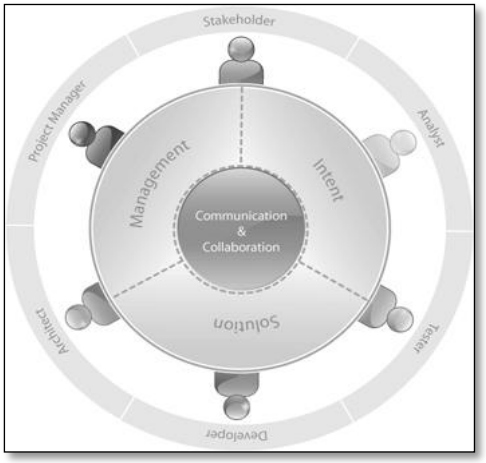

Figura 6: Papéis do OpenUp (OpenUP, 2010)

2.7.3 Disciplinas e Artefatos

Na Arquitetura de Métodos Unificada[15] (UMA), um produto de trabalho é um elemento de conteúdo que representa qualquer coisa usada, produzida ou modificada por uma tarefa. Uma disciplina é uma coleção de tarefas[16] que se relacionam a uma "área de interesse" maior em todo o projeto. O agrupamento de tarefas em disciplinas serve principalmente para ajudar a compreender o projeto dentro de uma visão tradicional em cascata[17]. Embora seja comum executar simultaneamente tarefas que pertençam a várias disciplinas (por exemplo, determinadas tarefas de requisitos são executadas sob a mesma coordenação de tarefas de análise e *design*), separar estas tarefas em disciplinas distintas é uma forma eficaz de organizar o conteúdo, tornando mais fácil a compreensão (OPENUP, 2010).

Abaixo, seguem as disciplinas do OpenUP que fornecem organização dos produtos de trabalho, e seus artefatos:

Disciplina: Arquitetura

Explica como criar uma arquitetura, a partir dos requisitos mais significantes. A arquitetura é construída na disciplina de Desenvolvimento.

- **Artefato:** Caderno de Arquitetura (*Architecture Notebook*)

  Este artefato descreve o contexto para o desenvolvimento do *software*. Ele contém as decisões, o raciocínio, as suposições, as explicações e as implicações da formação da arquitetura.

- **Disciplina:** Desenvolvimento

  Explica como projetar e implementar uma solução técnica que seja aderente à arquitetura e atenda aos requisitos.

---

[15] Arquitetura para a concepção, especificação e armazenamento de metadados de métodos e processos.

[16] Uma unidade de trabalho que um papel pode ser solicitado a executar.

[17] O primeiro modelo de desenvolvimento de *software* publicado. Originou-se de processos mais gerais de engenharia de sistemas (ROYCE, 1970) (SOMMERVILLE, 2007).

- **Artefato:** Design

  Este artefato descreve a realização da funcionalidade necessária para o sistema em termos de componentes e serve como uma abstração do código fonte.

- **Artefato:** Implementação

  Uma versão operacional de um sistema ou de uma parte do sistema que demonstre um subconjunto das capacidades a serem fornecidas no produto final é produzida.

- **Artefato:** Teste de Desenvolvedor

  Instruções que validam se os componentes individuais de *software* estão funcionando de acordo como o que foi especificado.

- **Disciplina:** Gestão de Projeto

  Explica como instruir, ajudar e suportar a equipe, ajudando-a a lidar com os riscos e obstáculos encontrados na construção de *software*.

- **Artefato:** Plano de Iteração

  Um plano detalhado que descreve os objetivos, as atribuições de trabalho e os critérios de avaliação para a iteração.

- **Artefato:** Plano de Projeto

  Este artefato registra todas as informações necessárias para gerenciar o projeto. Sua principal parte consiste em um plano genérico, contendo as fases e os marcos do projeto.

- **Artefato:** Lista de Itens de Trabalho

  Este artefato contém uma lista de todo trabalho agendado para ser feito dentro do projeto, bem como o trabalho proposto que pode afetar o produto neste ou em projetos futuros. Cada Item de

trabalho pode conter referências a informações que sejam relevantes para realizar o trabalho descrito no item.

- **Artefato:** Lista de Riscos

Este artefato é uma lista dos riscos conhecidos e existentes no projeto, classificados em ordem de importância e associados com as ações específicas de atenuação ou de contingência.

- **Disciplina:** Requisitos

Explica como elicitar, analisar, especificar, validar e gerenciar os requisitos para o sistema a ser desenvolvido.

- **Artefato:** Visão

Este artefato contém a definição da visão dos *stakeholders* a respeito do produto a ser desenvolvido, especificada em termos das principais características e necessidades dos *stakeholders*. Contém um esboço dos principais requisitos vislumbrados para o sistema.

- **Artefato:** Caso de Uso

Este artefato captura a sequência das ações executadas por um sistema que tenha um resultado de valor observável para aqueles que interagem com o mesmo.

- **Artefato:** Modelo de Caso de Uso

Este artefato captura um modelo das funções desejadas do sistema e de seu ambiente, e serve como um contrato entre o cliente e os desenvolvedores.

- **Artefato:** Glossário

Este artefato define termos importantes usados no projeto. Estes termos são a base para a colaboração eficaz com os *stakeholders* e outros membros da equipe.

- **Artefato:** Especificação de Requisitos Suplementares

  Este artefato captura todos os requisitos do sistema não capturados nos cenários ou nos casos de uso, incluindo requisitos de atributos de qualidade e não funcionais globais.

- **Disciplina:** Teste

Explica como fornecer *feedback* sobre a maturidade do sistema através do desenho, implementação, execução e avaliação dos testes.

- **Artefato:** Caso de Teste

  Este artefato é a especificação de um conjunto de entradas de teste, condições de execução e resultados esperados, identificados com a finalidade de fazer a avaliação de algum aspecto particular de um cenário.

- **Artefato:** Registro de Teste

  Este artefato coleta a saída capturada durante uma única execução de um ou mais testes para uma única realização do ciclo de teste.

- **Artefato:** Script de Teste

  Este artefato contém instruções passo a passo para realizar um teste, permitindo sua execução. Pode ter a forma de instruções textualmente documentadas que são executadas manualmente ou instruções interpretáveis pelo computador que permitem a execução automatizada de testes.

2.7.4 Considerações

O OpenUP destina-se a pequenas equipes que trabalham juntas no mesmo local. A equipe precisa se engajar em total interação face-a-face diariamente. Ela é composta por desenvolvedores, arquitetos, gerente de projeto e testadores. Eles tomam suas próprias decisões a respeito do que devem fazer, quais são as

prioridades e como melhor tratar as necessidades dos *stakeholders*. A organização deve suportar a equipe permitindo-lhes esta responsabilidade. Os membros da equipe colaboram intensamente. A participação dos *stakeholders* é crítica para o sucesso da implantação do OpenUP. Os integrantes da equipe participam de reuniões diárias para comunicar o *status* e possíveis dúvidas do projeto. Os problemas são tratados fora dessas reuniões.

O OpenUP foca na redução significativa dos riscos o mais cedo possível no ciclo de vida. Isto requer reuniões regulares de revisão dos riscos e rigorosas estratégias de atenuação. Todo o trabalho é relacionado, acompanhado e atribuído através da "Lista de Itens de Trabalho". Os membros da equipe usam este repositório único para registrar todas as tarefas necessárias e acompanhá-las. Isto inclui todas as solicitações de mudança, erros e pedidos dos *stakeholders*.

Casos de uso são desenvolvidos colaborativamente e usados para elicitar e descrever os requisitos, consequentemente, os membros da equipe devem desenvolver habilidades para escrevê-los bem. Os *stakeholders* são responsáveis por revisar e certificar que os requisitos estão corretos. Os requisitos arquiteturalmente significantes devem ser identificados e estabilizados na fase de Elaboração de forma que uma arquitetura robusta possa ser criada para ser o núcleo do sistema. Uma alteração em um requisito arquiteturalmente significante que tenha de ser tratada pode surgir posteriormente no desenvolvimento, mas o risco disto acontecer é significativamente reduzido a cada iteração da fase de Elaboração.

Testes são executados várias vezes por iteração, sempre que a solução for incrementada com o desenvolvimento de um requisito, uma mudança ou a correção de um erro. Estes testes ocorrem após os desenvolvedores terem construído a solução (que deve passar pelo teste de unidade) e a integrado no código base. Eles ajudam a garantir que uma construção estável seja criada e esteja sempre disponível à medida que o desenvolvimento progride. O OpenUP não inclui conteúdo para implantação e configuração de ambiente de desenvolvimento. Ele é focado em uma única equipe, e estas áreas são normalmente tratadas em nível organizacional ou empresarial (OPENUP, 2010).

3
Modelagem do Processo

Este capítulo apresenta a descrição do processo utilizado atualmente pela Fábrica de Software do LP&D. Descreve o novo processo de desenvolvimento de software, os métodos utilizados para levantar os requisitos necessários que o compõem e a ferramenta utilizada para modela-lo.

Processos são úteis porque imprimem consistência e estrutura a um conjunto de atividades. Estas características são importantes quando se sabe como fazer algo bem e quer garantir que outras pessoas o façam da mesma maneira (PFLEEGER, 2004). Para que este conhecimento seja persistido devem-se realizar atividades de análise e modelagem de processos, que envolvem o estudo e o desenvolvimento de um modelo abstrato dos processos. A análise esta relacionada com o estudo dos processos existentes em uma organização, no sentido de compreender os relacionamentos entre suas partes. A modelagem compreende a representação da abstração do processo utilizando uma notação específica (BPMN[18] e SPEM[19], por exemplo), sua descrição e documentação (SOMMERVILLE, 2007).

3.1 Análise do Atual Processo

Levantar os requisitos necessários para a concepção do processo de desenvolvimento de *software* da Fábrica de *Software* não foi tarefa fácil. Para

[18] *Business Process Modeling Notation*, ou notação para modelagem de processos de negócio, é o conjunto de conceitos e técnicas que visam à criação de um modelo com os processos de negócio existentes em uma organização.

[19] *Software Process Engineering Metamodel***,** é um metamodelo (uma linguagem) para processos de *software* (OMG, 2008).

realizá-lo foram feitas reuniões semanais com todos os membros da equipe de desenvolvimento e realizada a análise etnográfica.

3.1.1 Reuniões Semanais

As reuniões semanais tinham como objetivo capacitar todos os colaboradores da Fábrica de *Software* em técnicas para melhorar o desenvolvimento de *software* e criar um entendimento comum sobre os termos e práticas necessários para que o novo processo de desenvolvimento pudesse ser implantado.

Durante estas reuniões, guiadas pelo Prof. Rodrigo Martins Pagliares, eram discutidos temas como a elaboração de casos de uso de forma eficiente, o processo unificado de desenvolvimento de *software*, maneiras de medir a velocidade da equipe de desenvolvimento, etc. Estas, em conjunto com a analise etnográfica descrita na Seção 3.2.2, foram a base para a compreensão do que estava sendo utilizado como processo e possibilitaram delinear a base para a concepção de um novo processo, contribuindo para a definição e elaboração dos artefatos, tarefas, papéis, atividades e demais itens necessários.

3.1.2 Análise Etnográfica

Etnografia é uma técnica de observação que pode ser usada para compreender requisitos sociais e organizacionais (SOMMERVILLE, 2007). Nela, um observador imparcial se insere no ambiente de trabalho onde um sistema ou processo será utilizado. Ele observa o trabalho do dia-a-dia e anota as tarefas reais nas quais os participantes estão envolvidos. Seu valor está na ajuda que ela presta para descobrir requisitos implícitos que refletem os processos reais (mais ricos, complexos e dinâmicos), e não os formais, com os quais as pessoas de uma organização estão habituadas.

Esta analise foi empregada na Fábrica de *Software* do LP&D para que o novo processo de desenvolvimento pudesse ser elaborado de forma mais consistente com suas necessidades. Ela ocorreu entre os meses de fevereiro e maio de 2011, acompanhando diariamente o trabalho da equipe. Durante este período de tempo foram feitas entrevista informais para compreender o que era feito e de que forma eles executavam suas tarefas.

3.1.3 Descrição do Atual Processo

A Fábrica de *Software* do Laboratório de Pesquisa e Desenvolvimento não conta com um processo formal de desenvolvimento de *software*. Entretanto, existem passos seguidos pela equipe de desenvolvimento que informalmente caracterizam a maneira como desenvolvem *software*. Os passos no atual processo estão descritos abaixo e são exibidos na Figura 7:

- **Entrevista:** obter dos interessados no projeto entendimento sobre o que deve ser feito, capturar requisitos, esclarecer dúvidas e obter *feedback* sobre o que já foi feito.

- **Elicitação de Requisitos:** detalhar os requisitos encontrados na entrevista.

- **Prototipação:** criar um protótipo funcional através dos requisitos levantados na entrevista e detalhados na elicitação.

- **Validação dos Requisitos:** validar requisitos através dos protótipos construídos.

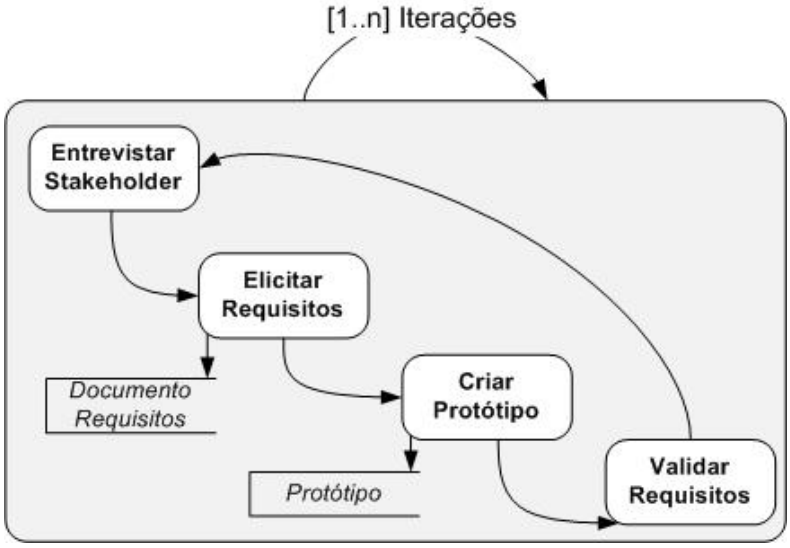

Figura 7: Atual Processo de Desenvolvimento da Fábrica de *Software*

Estes quatro passos são executadas de forma sequencial, o que caracteriza a influência do modelo de desenvolvimento em cascata. Porém, não se pode dizer

que o atual processo segue a risca tal modelo de processo, pois cada etapa não é executada uma única vez durante o ciclo de vida do projeto. A cada iteração, requisitos levantados na entrevista e descritos na etapa de elicitação são implementados em um protótipo e validados posteriormente. A documentação produzida é mínima e ineficaz. Mínima porque durante todo o ciclo de vida do projeto o único documento produzido é o documento de requisitos. Ineficaz porque, apesar de existir não é consultada pelos desenvolvedores. A gestão de projetos atual segue a linha tradicional descrita pelo Guia PMBOK® (PMBOK, 2008) e se limita ao uso da ferramenta de gerência de projetos Project da Microsoft, não produzindo nenhum documento adicional.

3.2 Novo Processo de Desenvolvimento

É fato que o atual processo de desenvolvimento de *software* acima descrito é muito limitado. Ele precisa ser desenvolvido com o intuito de prover melhor organização e controle sobre o que é produzido na Fábrica de *Software*. O novo processo possui a característica de não ser muito prescrito, evitando assim torna-se pesado. Contém apenas a documentação mínima necessária para apoiar os projetos, é ágil e suporta a constante evolução e adaptação. Esta seção resume os principais aspectos do processo de desenvolvimento de *software* concebido para atender as necessidades da Fábrica de *Software* do LP&D.

3.2.1 Modelagem: Eclipse Process Framework

Para modelar o processo de desenvolvimento de *software* da Fábrica de *Software* do LP&D foi utilizado o Eclipse Process Framework[20] (EPF), um projeto open source[21] da Eclipse Foudation[22]. Ele é um *framework* de autoria, customização e

[20] Disponível em http://www.eclipse.org/epf.

[21] Método de desenvolvimento de *software* que se baseia no poder da distribuição de processos e na sua transparência. Promete uma melhor qualidade e confiabilidade, mais flexibilidade, menores custos e fim ao código proprietário e fechado (OPEN SOURCE, 2011).

[22] *Eclipse Foundation* é uma comunidade *open source*, cujos projetos são focados em construir uma plataforma de desenvolvimento aberta composta por *frameworks* extensíveis, ferramentas a construção, implantação e gerenciamento de *software* em todo o ciclo (ECLIPSE, 2011).

publicação de processos de desenvolvimento de *software* que utiliza SPEM[23] (Seção 6.1). A arquitetura definida pelo EPF tem entre seus principais objetivos facilitar a adaptação, personalização, extensão, evolução e publicação de processos (PAULA FILHO, 2009). A Figura 8 exibe algumas funcionalidades do EPF e abaixo é descrito seus dois objetivos principais (HAUMER, 2007):

- Fornecer para os praticantes de desenvolvimento um conhecimento que permita a busca, gerência e distribuição do conteúdo. Esse conteúdo pode ser licenciado, adquirido, e deve acomodar seu próprio conteúdo, por exemplo, definição de métodos, princípios, melhores práticas, procedimentos internos e regulamentações.

- Fornecer as capacidades de engenharia de métodos, oferecendo suporte aos engenheiros de métodos e aos gerentes de projeto para que possam selecionar, customizar, e rapidamente recolher processos para o desenvolvimento de projetos. O EPF Composer fornece catálogos de processos pré-definidos para as situações típicas de projeto que podem ser adaptadas às necessidades de cada indivíduo. Ele também fornece processos construídos por blocos chamados capability patterns que representam as melhores práticas de desenvolvimentos para disciplinas específicas, tecnologias, ou estilos de desenvolvimento.

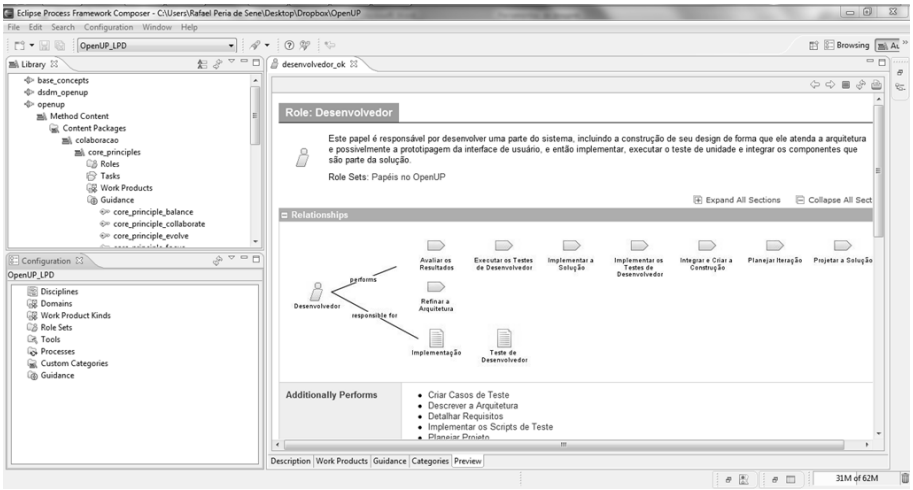

Figura 8: Funcionalidades do EPF Composer

[23] *Software Process Engeneering Metamodel*, descreve um metamodelo para processos de *software*, ou seja, uma linguagem para descrição de processos de *software* (PAULA FILHO, 2009).

3.2.2 Descrição do Novo Processo

Processos de desenvolvimento de *software* são importantes pois contribuem para que se possa capturar experiências e passá-las à diante. Eles ajudam a manter um nível de consistência e qualidade nos produtos e serviços produzidos por pessoas distintas. Sua estrutura orienta ações, permitindo examinar, enteder, controlar e aprimorar as atividades que o compõem. Eles podem ser descritos e modelados de maneira flexível, utilizando textos, figuras ou alguma combinação destes, levando-se em consideração suas atividades, com seus critérios de entrada e saída e a ordem em que devem ocorrer, os recursos que utiliza, as restrições a que está sujeito, seus subprocessos e os produtos intermediários e finais gerados (PFLEEGER, 2004).

Nesta monografia, o processo de desenvolvimento de *software* da Fábrica de *Software* do LP&D é descrito em alto nível, contendo os itens necessários para sua compreensão. Detalha-lo completamente aqui prolongaria esta monografia em dezenas de páginas, tornado-a deveras cansativa. Os detalhes sobre cada item que o compõem podem ser encontrados em (PROCESSO LP&D, 2011).

O processo da Fábrica de *Software* possui *work breakdown structure* composta por quatro fases: Concepção (Figura 9), Elaboração (Figura 10), Construção (Figura 12) e Transição (Figura 13). Estas são as mesmas fases prescritas pelo Processo Unificado (Seção 2.2) e pelo OpenUP (Seção 2.5) e possuem, respectivamente, os seguintes marcos: estudo de viabilidade e aprovação do projeto, consolidação da arquitetura, construção do *software* e implantação do que foi construído em ambiente de desenvolvimento.

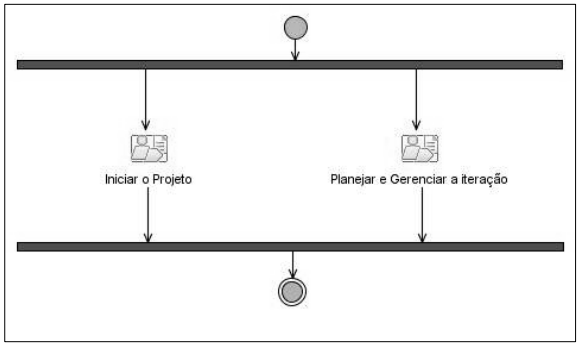

Figura 9: Diagrama de Atividades da Fase de Concepção

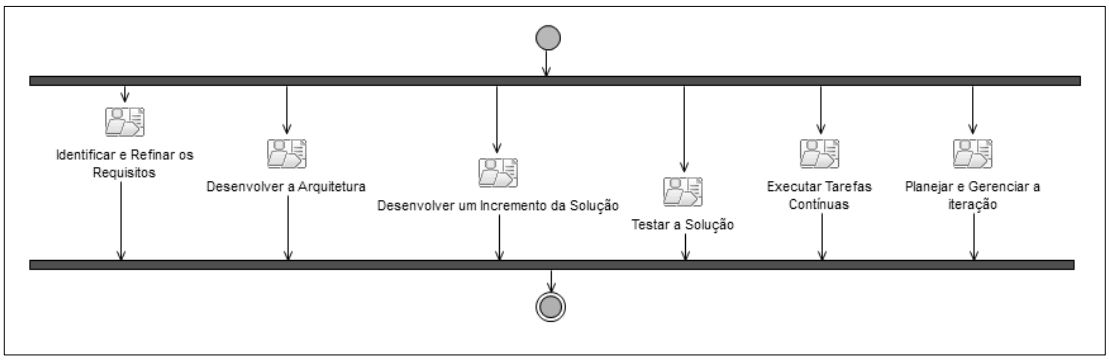

Figura 10: Diagrama de Atividades da Fase de Elaboração

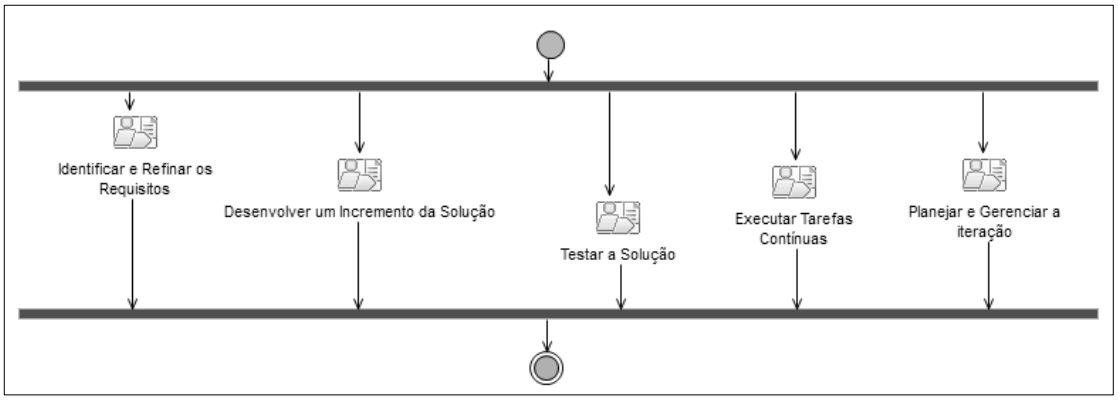

Figura 11: Diagrama de Atividades da Fase de Construção

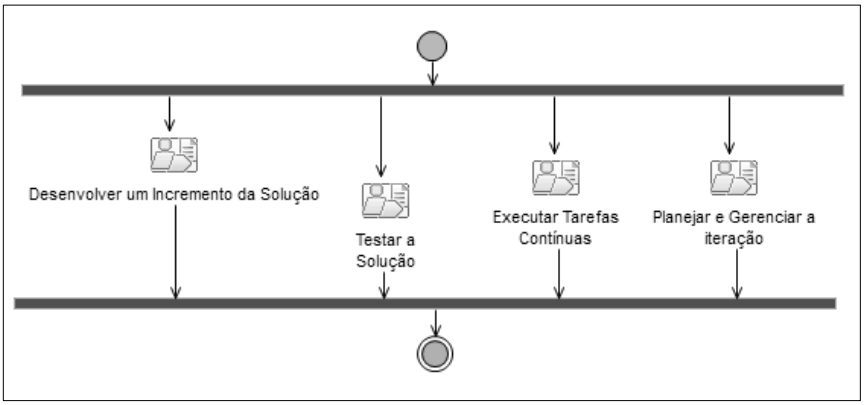

Figura 12: Diagrama de Atividades da Fase de Transição

Cada fase, com exceção da Concepção que compreende apenas uma iteração, pode ser realizada em um número de iterações que varia de apenas uma até a quantidade necessária, dependendo das características do projeto sendo executado. É composta por atividades, que englobam as tarefas prescritas para que um projeto que instancie o processo possa desenvolver-se de maneira organizada e estruturada. As tarefas que compõem cada atividade em cada

uma das fases são exibidas nas Figuras 13, 14, 15 e 16. Cada tarefa pertence a uma disciplina específica. As disciplinas são as mesmas que compõem o OpenUP (Arquitetura, Desenvolvimento, Requisitos, Gestão e Testes), com acréscimo da definição da disciplina de Gestão de Configuração e Mudança, descrita no trabalho de conclusão de curso intitulado "*Definição das disciplinas de Gestão de Configuração e Testes para o Processo de Desenvolvimento de Software do LP&D*", do discente do curso de Bacharelado em Ciência da Computação da Universidade Federal de Alfenas Rômulo Silva Campos.

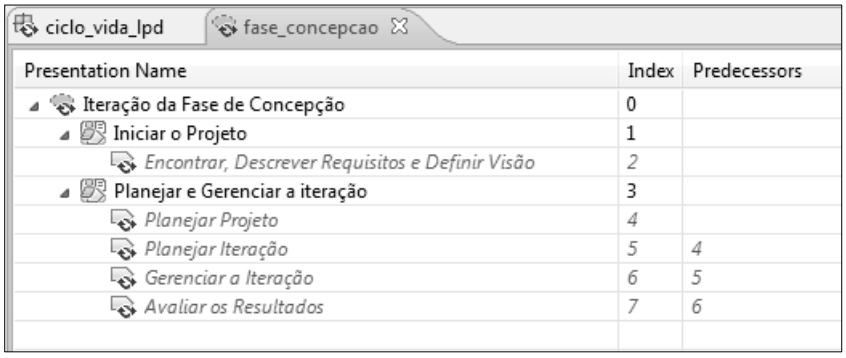

Figura 13: WBS da Fase de Concepção

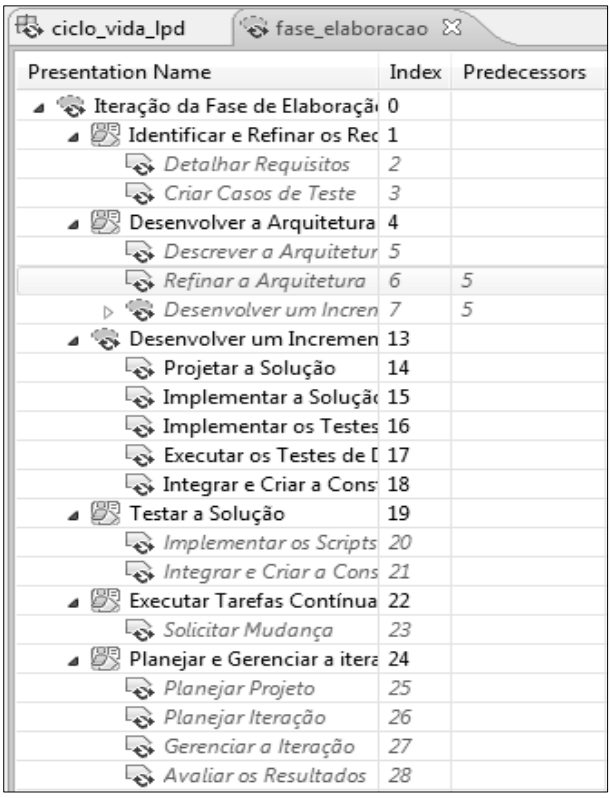

Figura 14: WBS da Fase de Elaboração

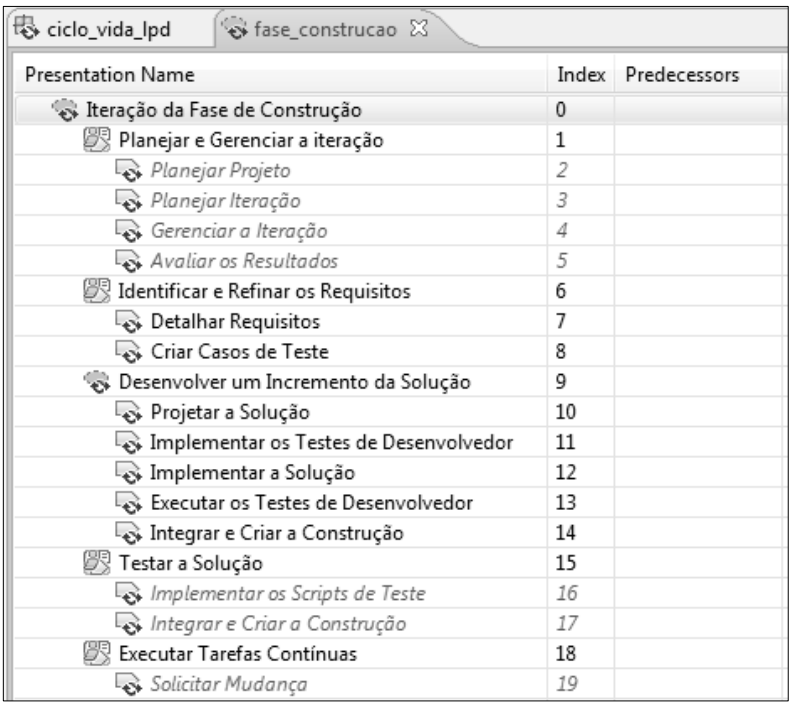

Figura 15: WBS da Fase de Construção

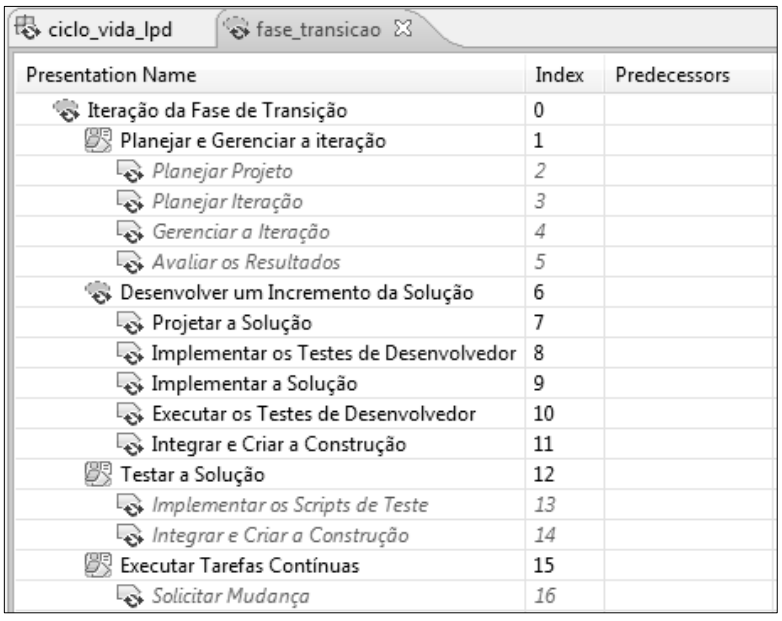

Figura 16: WBS da Fase de Transição

Cada tarefa é executada por um ator: Analista de Negócios, Gerente de Projetos, Analista de Teste, Desenvolvedor, *Stakeholder*. A Figura 17 exibe um exemplo de um ator com suas atribuições.

Figura 17: Gerente de Projetos e suas atribuições

As tarefas possuem passos que devem ser seguido para executa-las com sucesso, a Figura 18 exibe um exemplo dos passos necessários para realizar a tarefa "Encontrar, Descrever Requisitos e Definir Visão".

Figura 18: Passos da Tarefa Encontrar, Descrever Requisitos e Definir Visão

Toda tarefa produz/consome produtos de trabalho. A Seção 6.2 apresenta um exemplo de produto de trabalho denominado "Visão", completamente preenchido, escrito utilizando-se o modelo definido pelo processo.

Como dito anteriormente, o processo fora aqui descrito em alto nível e os detalhes podem ser encontrados no sítio onde ele está publicado.

3.2.3 Implantação do Processo: primeiros passos

Existem hoje em andamento dois projetos-piloto que estão instanciando o processo modelado. Ambos os projetos são voltados para o desenvolvimento de portais de conteúdo para setores específicos da Universidade Federal de Alfenas.

O uso do processo por estes projetos tem como objetivo encontrar pontos de inconsistência no que fora modelado, bem como avaliar a sequencia de atividades, as tarefas e os produtos de trabalho prescritos.

Atualmente os dois projetos encontram-se na fase de Elaboração do processo. Os dados registrados destes dois projetos-pilotos serão utilizados para aperfeiçoá-lo e evoluí-lo.

4
Conclusões

Este capítulo apresenta as conclusões obtidas a partir dos trabalhos realizados e são discutidas as propostas futuras que poderão ser realizadas acerca deste tema.

Processos de *software* não são "balas de prata". Segui-los não implica que o *software* será desenvolvido de forma consistente, dentro de prazos e custos estimados e contemplará características de qualidade desejáveis. Segue-se um processo porque ele imprime consistência e estrutura a um conjunto de atividades necessárias para o desenvolvimento de *software*.

Hoje, a Fábrica de *Software* do LP&D não possui um processo formal de desenvolvimento de *software* como arcabouço para os projetos de *software* que ela desenvolve. Seus colaboradores seguem um processo de desenvolvimento informal, mas funcional, caracterizado pela criação evolutiva de protótipos, de acordo com os requistos estabelecidos pelos seus clientes. Porém, devido a crescente demanda por desenvolvimento de *software*, a Fábrica de *Software* deve estruturar-se para produzir *software* de qualidade de maneira efetiva.

A definição de um processo de desenvolvimento é o primeiro passo para isto, e é o que foi feito neste trabalho de conclusão de curso, utilizando-se o Processo Unificado Aberto (OpenUP) como base para conceber o processo de desenvolvimento de *software* da Fábrica de *Software* do LP&D. Para isto, foram necessários vários meses de investigação utilizando-se da análise etnográfica para compreender de forma eficiente quais os reais requisitos o processo que estava sendo criado deveria atender. Paralelamente a essa imersão no ambiente de desenvolvimento, foram realizadas reuniões semanais para treinamento da equipe em métodos de desenvolvimento de *software* e a apresentação dos principais conceitos necessários para a compreensão do processo em construção. Capturados os requesitos, adaptava-se o OpenUP para satisfazê-los e modelava-os utilizando o *software open source* EPF Composer.

Através das investigações, concepção e modelagem tem-se hoje a primeira versão do processo de desenvolvimento de *software* para a Fábrica de *Software* do LP&D. Um processo leve e pouco prescritivo, iterativo e incremental, com um ciclo de vida estruturado e ágil, focado na natureza colaborativa do desenvolvimento de *software*. Este está em fases de testes, sendo instanciado por dois projetos-piloto como forma de avaliar sua consistência, encontrar pontos onde seja necessário melhora-lo e avaliar sua viabilidade de execução.

A partir de agora o processo tende a evoluir e a adaptar-se constantemente, de forma a tornar-se cada vez mais eficiente. A melhoria contínua está presente no dia a dia. Esta é uma tarefa de longa duração, que já se iniciou com a elaboração da segunda versão do processo e será continuada em trabalhos futuros juntamente com a adequação do processo para que os requisitos do CMMI sejam atendidos.

5
Apêndice

5.1 Apêndice 1: SPEM

O SPEM (*Software Process Engineering Metamodel*) foi criado pelo Object Management Group (OMG), para ser um padrão de alto nível para processos usados em desenvolvimento de *software* (HENDERSON e SELLERS, 2005). Inicialmente, o SPEM foi criado como um meta-modelo independente e mais tarde foi estendido como um perfil da UML, ou seja, tornou-se possível a autores de processo transformar concepções criadas orientadas a processo em concepções orientadas a modelo (HENDERSON e SELLERS, 2005; NARDINI 2008; PAULA FILHO 2009). Atualmente o meta-modelo encontra-se na versão 2.0, faz uso de outras especificações da OMG e reutiliza a estrutura da UML (*Unified Modeling Language*) 2 sempre que possível (OMG, 2008).

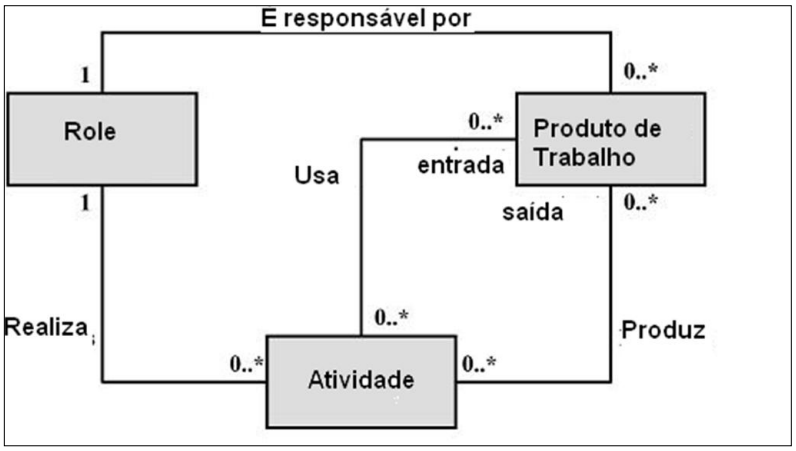

Figura 19: Estrutura Central de um Processo como definida pelo SPEM (RANSIN, 2008)

Ele trata um processo de desenvolvimento de *software* como uma colaboração de entidades ativas (papéis) direcionadas à realização de operações específicas, chamadas atividades, em um conjunto de artefatos, nomeados

produtos de trabalho, até que estes artefatos são levados a um estado bem definido e declarado completo. O meta-modelo possui então uma estrutura central que representa um processo de *software* através de papéis, os produtos de trabalho pelos quais cada papel é responsável e as atividades que estes papéis realizam nestes produtos (RANSIN, 2008). Essa organização geral é mostrada na Figura 17, ressaltando que a estrutura completa do SPEM é muito mais complexa que esta estrutura central.

6
Referências Bibliográficas

Agile, *Manifesto for Agile Software Development,* 2010. Disponível em <http://agilemanifesto.org. Acessado em 01-out-2010>.

AgileData, *Techniques for Successful Evolutionary/Agile Database Development,* 2011. Disponível em <http://www.agiledata.org/. Acessado em 11-mai-2011>.

Baetjer, Ir., H., *Software as Capital,* IEEE Computer Society Press, 1998. p.85.

Beck, K. *Extreme Programming explained.* Addison-Wesley. Boston, 2000.

Booch, G., *Object-Oriented Analysis and Design.* Second Edition, Benjamin Cummings, 1994.

Brinkkemper, S. *Method Engineering: Engineering of Information Systems Development Methods and Tools*, In: Information and *Software* Technology, Volume 38, fourth edition, 1996.

Businesscase, *Ciclo Estratégico da Informação*, 2011. Disponível em <http://www.businesscase.com.br>. Acessado em 22-jan-2011.

Cohn, M. *Desenvolvimento de Software com Scrum: Aplicando Métodos Ágeis com Sucesso.* Primeira Edição. Bookman, 2011.

CMMI, *CMMI for Development, Version 1.2, CMU/SEI-2006-TR-008. Development Methods and Tools*, In: Information and *Software* Technology, vol. 38, fourth edition, 2006.

Eclipse, *Eclipse Foundation,* 2010. Disponível em <http://epf.eclipse.org>. Acesso em 14-nov-2010.

Henderson-Sellers, B., Gonzalez-Perez, C., *A comparsion of four process metamodels and the creation of a new generic standard*, Univesity of Technology, Sidney, Australia. Publicado em: Information and *Software* Technology nº 47, páginas 49-65, 2005.

Haumer, P, *Overview to Eclipse Framework Composer,* Eclipse Foundation, 2007.

IEEE Software Engeneering Collection on CD-ROM. IEEE, New York, 2003.

Jacobson, I. *Object-Oriented Software Engineering,* Addison-Wesley, 1992.

Johnson, Ralph E., Frameworks = (Components + Patterns), *Communications of the ACM*, vol. 40 n. 10, pp. 39-42, 1997.

Kruchten, Phillippe. *Introdução ao RUP: Rational Unified Process*, 1ª edição, Ciência Moderna, 2003.

Larman, Craig. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo*, 3ª Edição, Bookman, 2007.

MountainGoat, *Moutain Goat Software,* 2011. Disponível em <http://www.mountaingoat*software*.com>. Acesso em 20-abr-2011.

Nardini, E., Omincini, A., Denit, E., Molesini, A., *SPEM on test: the SODA case study,*SAC'08, páginas 16-20, Março, 2008, Fortaleza, Ceará, Brasil.

OpenUP, *Open Unified Process.* Versão 1.5.0.4. 10-08, 2010. Disponível em <http://epf.eclipse.org>. Acessado 15-set-2010.

OpenSource, *The Open Source Initiative,* 2011. Disponível em <http://www.opensource.org>. Acessado em 20-abr-2011.

OMG, *Software & System Process Engineering Metamodel Specification*, Versão 2.0, 2008.

Paula Filho, Wilson de Pádua. *Engenharia de Software: Fundamentos, Métodos e Padrões.* 3ª edição, LTC, Rio de Janeiro, 2009.

Pall, Gabriel A. *Quality Process Management.* Englewood Cliffs, N.J.: Prentice Hall, 1987.

Pfleeger, S.L. and Atlee, J. M., *Software Engineering: theory and practice*, third edition, Prentice Hall, 2005.

Pfleeger, Shari Lawrance. *Engenharia de Software: teoria e prática,* 2ª edição, Prentice Hall, 2004.

PMBOK 2008. *Project Management Body of Knowledg*. Project Management Institute, 4ª edição, 2008.

Pressman, Roger S. *Software Engineering: a practitioner's approach.* seventh edition, McGraw-Hill, 2010.

Pressman, Roger. S., *Engenharia de Software*, 6ª edição McGraw-Hill, 2006.

ProcessoLP&D, *Processo de Desenvolvimento de Software da Fábrica de Software do Laboratório de Pesquisa e Desenvolvimento,* 2011. Disponível em: <*http://www.bcc.unifal-mg.edu.br/~pagliares/openup/fabrica/Publish_final_v59 / index.htm*>.

Ramsin, R., Paige, F. *Process-Centered Review of Object Oriented Software Development Methods*, ACM Computing Survey, Vol 40, N 1, artigo 3, 2008.

Royce, W. W. "*Managing the development of large software systems: concepts and techniques*". Proc. IEEE WESTCON, Los Angeles CA: IEEE Computer Society Press, 1970.

Rumbaugh, J., *et al, Object-Oriented Modeling and Design,* Prentice-Hall, 1991.

ScrumAlliance, *Transforming the World of Work*, 2010. Disponível em <http://www.scrumalliance.org>. Acessado em 14-nov-2010.

Sommerville, I. *Engenharia de Software*, 8ª edição, Pearson Addison-Wesley, 2007.

7
Anexo

7.1 Anexo 1: Documento Visão “Portal CEAD”

Fábrica de *Software* – LP&D

Conteúdo

Histórico **78**
Histórico de Revisão 78
Revisão do Documento 78
Aprovação do Documento 78
1. Introdução **78**
1.1 Objetivo 78
1.2 Escopo 79
2. Oportunidade de Negócio **79**
2.1 Histórico 79
2.2 Posicionamento 79
3. Atores **79**
4. Solução Proposta **80**
4.1 Características Principais – Requisitos Funcionais 80
4.1.1 Características Essenciais 80
4.1.2 Características Importantes 81
4.1.3 Características Desejável 81
4.2 Qualidades Sistêmicas Primordiais – Requisitos não funcionais 81
Confiabilidade e Disponibilidade 81
Qualidade de serviço 82
Usabilidade 82
Manutenibilidade. 82
5. RISCOS 82
6. RESTRIÇÕES 82
6.1 Processo de Desenvolvimento e Restrições de Equipe 82
6.2 Restrições Tecnológicas 82
6.3 Restrições de Entrega e Implantação 83

TABELA 1: ATORES 79

Histórico

Histórico de Revisão

Data	*Versão*	*Autor*	*Descrição*
13/05/2011	1.0	Mariane, Raquel	Criação inicial do documento
16/05/2011	1.0	Mariane, Raquel	Adição dos Requisitos Funcionais
17/05/2011	1.0	Mariane, Raquel	Adição dos Requisitos Não-Funcionais

Revisão do Documento

Data	*Versão*	*Nome do Revisor*	*Informação para Contato.*

Aprovação do Documento

Data	*Versão*	*Nome do Avaliador*	*Informação para Contato.*

1. Introdução

1.1 Objetivo

Este documento define o escopo do projeto para o Portal CEAD. Ele estabelece a necessidade de negócio do Portal CEAD e resume os requisitos em alto nível de abstração, necessários para satisfazer as necessidades de negócio especificadas. Este documento não é uma descrição completa de requisitos, mas fornece uma visão global para um conjunto de requisitos do sistema mais detalhado. O principal objetivo deste

documento é fornecer prioridades entre os requisitos mais importantes, considerações, características e metas gerais.

1.2 Escopo

Este portal irá disponibilizar as informações gerais do CEAD, tais como cursos de graduação e pós-graduação oferecidos à distância, polos dos cursos, editais de contratação e processos seletivos, notícias relacionadas ao CEAD e ao ensino à distância em geral, eventos tais como oficinas, palestras, congressos, dentre outras informações.

O portal também disponibilizará um espaço para a ferramenta virtual de aprendizagem Moodle, com vídeos para professores e alunos, auxiliando a utilização do ambiente, tutoriais que mostrem passo-a-passo a manipulação das funcionalidades, dentre outros meios de suporte à ferramenta.

2. Oportunidade de Negócio

2.1 Histórico

O desenvolvimento do Portal do CEAD é importante em vários contextos. Na divulgação do Centro de Educação Aberta e à Distância da Unifal, na disponibilização de notícias, eventos e oficinas oferecidas pelo CEAD, e no auxílio à manipulação do ambiente Moodle.

2.2 Posicionamento

O Portal do CEAD é um produto já existente e no contexto deste projeto serão realizadas várias melhorias de modo a atender com mais qualidade as necessidades dos usuários. Trata-se de um sistema de infra-estrutura compartilhada com ambiente virtual Moodle, como um dos meios de acesso à ferramenta. A nova versão a ser desenvolvida irá substituir a versão existente, provendo melhorias como acessibilidade de informações e aparência. Tais melhorias irão contribuir para a melhor divulgação dos produtos e serviços do Centro de Educação Aberta e à Distância da Unifal-MG.

3. Atores

Estas são as funções exercidas por pessoas que interagem com o sistema.

Nome do Ator	Descrição
Administrador do Portal	Responsável por atividades gerais de manutenção tais como atualização de conteúdo, resolução de problemas, dentre outros.
Usuário externo	Acessará o portal via web com diferentes objetivos, tais como acessar informações do ensino à distância, acessar o espaço do ambiente moodle e a própria ferramenta moodle.

Tabela 3: Atores

4. Solução Proposta

4.1 Características Principais – Requisitos Funcionais

Nesta seção, classificaremos as características principais do Portal CEAD dentro de três categorias. Características Essenciais não podem ficar de fora. Características Importantes podem estar ausentes, embora não seja desejável. Características Não Essenciais são aquelas que não estão claras na primeira versão.

4.1.1 Características Essenciais

É essencial que o portal disponibilize diversas informações incluindo as seguintes categorias:

- Notícias: notícias gerais relacionadas ao CEAD, incluindo novos editais, eventos em destaque, novidades do ensino à distância e do ambiente Moodle. As notícias mais atuais serão exibidas na página inicial. Além disso, na página inicial haverá um direcionamento para as notícias menos atuais.
- Contato e localização: informações tais como localização do CEAD, nome, cargo, e-mail e ramal de cada membro da equipe, visando facilitar a comunicação dos usuários com os responsáveis por cada setor do CEAD. Além disso, o Portal disponibilizará um formulário online por meio do qual o usuário poderá enviar dúvidas e sugestões para os responsáveis de cada setor.
- Espaço Moodle: na página inicial o usuário poderá ser redirecionado para um espaço reservado ao ambiente virtual de ensino e aprendizagem Moodle, que disponibilizará diversos serviços:
    - Login: campos para login e senha de acesso ao ambiente.
    - Informações gerais: informações gerais sobre o ambiente tais como versão atual, melhorias, recursos disponíveis, métodos de acesso, dentre outros.
    - Perguntas frequentes: visando facilitar a utilização do ambiente por alunos e professores, o espaço Moodle disponibilizará as soluções para as principais dúvidas encontradas pelos usuários.
    - Vídeo-aulas e tutoriais: serão disponibilizados vídeo-aulas e tutoriais referentes à utilização do ambiente, visando facilitar ainda mais a utilização do Moodle pelas diferentes categorias de usuário.
    - Contato: informações como nome, email e ramal dos responsáveis pelo suporte técnico de cada categoria de usuário, além de um formulário online para envio de dúvidas e sugestões sobre o ambiente.
- Cursos de Graduação: serão disponibilizadas informações sobre cursos de Graduação à Distância oferecidos pelo CEAD, tais como objetivos dos cursos, polos dos cursos, perfil do egresso, dinâmicas curriculares, dentre outros.

- Curso de Pós-Graduação: serão disponibilizadas informações sobre cursos de Graduação à Distância oferecidos pelo CEAD, tais como objetivos dos cursos, polos dos cursos, perfil do egresso, dinâmicas curriculares, dentre outros.

4.1.2 Características Importantes

É importante que o portal disponibilize diversas informações incluindo as seguintes categorias:

- Editais: editais de abertura de concurso para tutores à distância, processos seletivos para os cursos à distância, processos seletivos para estagiários remunerados. Estarão acessíveis tanto os editais abertos quanto aqueles já encerrados, bem como seus resultados.
- Eventos: eventos realizados pelo CEAD tais como capacitações, palestras e oficinas. Irá disponibilizar também informações sobre eventos importantes relacionados com o ensino à distância e ao ambiente virtual de aprendizagem Moodle.
- CEAD: serão disponibilizadas informações sobre o Centro de Educação Aberta e à Distância da Unifal-MG, bem como seus membros participantes. Além disso, serão disponibilizados links úteis relacionados ao CEAD, tais como Universidade Aberta do Brasil (UAB), Web-Conferência da RNP, dentre outros.

4.1.3 Características Desejáveis

É desejável que o portal disponibilize diversas informações incluindo as seguintes categorias:

- Direcionamento para redes sociais: haverá um espaço de direcionamento para redes sociais incluindo Facebook e Twitter do CEAD, para que, caso seja do interesse do usuário, este possa acompanhar a divulgação de informações através destes meios.
- CEAD Online: os usuários poderão entrar em contato com o suporte técnico do CEAD através de serviços de mensagens instantâneas como Messenger e Skype.
- Polos: serão disponibilizadas informações sobre cada polo tais como: localização, contatos, cursos oferecidos, fotos, estrutura, dentre outros.

4.2 Qualidades Sistêmicas Primordiais – Requisitos não funcionais

Nesta seção, identificamos os principais requisitos de nível de serviço para o Portal CEAD. Esta não é uma lista completa e possui como objetivo capturar os requisitos mais importantes através de uma perspectiva de negócio.

Confiabilidade e Disponibilidade

O Portal CEAD deverá estar sempre disponível aos seus usuários, podendo ser acessado tanto no ambiente interno da universidade quanto no ambiente externo. Caso ocorra uma falha ou queda no servidor, este deverá se recuperar prontamente.

Qualidade de serviço
O Portal CEAD deverá estar sempre atualizado com relação às notícias, editais, resultados, eventos, dentre outros, refletindo a realidade do Centro de Educação Aberto e à Distância.

Usabilidade
O Portal CEAD deverá oferecer os seus serviços de modo a facilitar o acesso a todo tipo de informação desejada. Para isso serão utilizados diversos recursos tais como ícones de fácil identificação, menus interativos, figuras ilustrativas, dentre outros.

Manutenibilidade
O Portal CEAD deverá ser desenvolvido de forma a facilitar futuras manutenções tanto da própria equipe de criação quanto outras equipes diferentes.

5. Riscos
Alguns riscos podem atrapalhar o bom andamento do projeto:

- Escolha da ferramenta para gerência de conteúdo do Portal do CEAD.
- Mudanças drásticas de requisitos já levantados.
- Abandono dos membros da equipe de desenvolvimento por motivos diversos.
- Falta de recursos relacionados à infraestrutura tais como máquinas, salas, mesas e acesso à internet.

6. Restrições

6.1 Processo de Desenvolvimento e Restrições de Equipe
Para o desenvolvimento deste projeto será utilizado o processo da Fábrica de *Software* LP&D, que representa uma adaptação do processo ágil OpenUp, que pode ser acessado através do site:

http://www.bcc.unifal-mg.edu.br/~pagliares/openup/fabrica/Publish_final_v59/.

Com relação à gestão do projeto, será utilizado o método ágil SCRUM (http://www.scrumalliance.org/)

A equipe será formada por quatro integrantes:

- Desenvolvedor WEB: responsável pelo desenvolvimento do Portal.
- Web-Design: responsável pelo desenvolvimento de interfaces amigáveis ao usuário relacionadas ao Portal.
- Engenheiro de teste: responsável pela realização dos testes relacionados às funcionalidades desenvolvidas para o Portal.
- Gerente de redes: responsável pela manutenção do servidor.

6.2 Restrições Tecnológicas
Algumas ferramentas serão utilizadas para o desenvolvimento do projeto:

- Joomla – Gerenciador de Conteúdo para desenvolvimento do Portal.
- Photoshop Express – *Software* para edição de imagens.
- Selenium – *Software* para geração de scripts de testes automatizados.
- Hudson – *Software* para integração contínua.
- Mantis – *Software* para gestão de defeitos no Portal.

6.3 Restrições de Entrega e Implantação

O Portal CEAD deverá ser implantado em um dos servidores da Unifal-MG, responsável por fornecer o acesso contínuo dos usuários. O acesso ao Portal CEAD deverá ser feito por computadores com acesso à internet.